robotron

Systemunterlagendokumentation

MOS K1520

Anwenderdokumentation

Zusatzsoftware III

unter dem Betriebssystem SCPX 1526

Stand: 31.12.85

Anleitung fuer den Bediener fuer die "Zusatzsoftware III unter dem Betriebssystem SCPX 1526"

VEB Robotron Buchungsmaschinenwerk

Karl-Marx-Stadt 1985

Die vorliegende 1. Auflage der - Dokumentation "Zusatzsoftware III unter dem Betriebssystem SCPX 13526" - entspricht dem Stand Dezember 1985 und unterliegt nicht dem Aenderungsdienst.

Nachdruck, jegliche Vervielfaeltigung oder Auszuege daraus sind unzulaessig.
Die Dokumentation wurde durch ein Kollektiv des VEB Robotron Buchungsmaschinenwerk Karl-Marx-Stadt ausgearbeitet.
Im Interesse einer staendigen Weiterentwicklung werden die Leser gebeten, dem Herausgeber ihre Vorschlaege bzw. Hinweise zur Verbesserung mitzuteilen.

Anmerkung:

Die Anwenderdokumentation zum SCP-System besteht aus:

- Anleitung fuer den Bediener SCP 1520
- Anleitung fuer den Programmierer SCP 1520
  Teil I und Teil II (Sprachbeschreibung ASM)
- Anleitung fuer den Systemprogrammierer SCP 1520
- Hardwarebeschreibung
- Anwenderdokumentation BASIC-Interpreter
- Anwenderdokumentation BASIC-Compiler
- Anwenderdokumentation C-Compiler
- Anwenderdokumentation PASCAL-Compiler
- Anwenderdokumentation FORTRAN-Compiler
- Anwenderdokumentation Textverarbeitungssystem TP
- Anwenderdokumentation Installierungs-Programm fuer TP
- Anwenderdokumentation KOMBO-Druck
- Schulungshandbuch fuer das Textverarbeitungssystem TP
- Anwenderdokumentation Kalkulationsprogramm
- Anwenderdokumentation Datenerfassungssystem DAT
- Anwenderdokumentation Datenbanksystem REDABAS
- Anleitung fuer den Systemprogrammierer PASCAL/M2
- Zusatzsoftware I
- Zusatzsoftware II
- Zusatzsoftware III

VEB Robotron-Buchungsmaschinenwerk
Karl-Marx-Stadt
Software-Zentrum
9010 Karl-Marx-Stadt/PSF 129

INHALTSVERZEICHNIS

		Seite
1.	Treiber-Unterprogramme	6
1.1.	Treiber fuer Kassettenmagnetbandgeraet K 5200 KMBG 1520 (SCPX)	6
1.1.1.	Aufgabe des Programms	6
1.1.2.	Aufrufsequenz des Treibers	6
1.1.3.	Programmbedingungen	8
1.1.4.	Erlaeuterung der Funktionen	8
1.1.4.1.	Lesen 1 Block	8
1.1.4.2.	Schreiben 1 Block	9
1.1.4.3.	Schreiben Bandmarke	9
1.1.4.4.	Vor-/Ruecksetzen 1 Block	9
1.1.4.5.	Vor-/Ruecksetzen 1 Bandmarke	9
1.1.4.6.	Schnelles Vor- bzw. Rueckspulen	9
1.1.4.7.	Ver- bzw. Entriegeln KMB	9
1.2.	Treiber fuer den Anschluss der Schreib-Leseeinheit SLE 1520 (SCPX)	10
1.2.1.	Hardware	10
1.2.1.1.	Schreibleseeinheit und Anschlussteuerung	10
1.2.1.2.	Technische Daten	10
1.2.1.3.	Fehlererkennung	11
1.2.1.4.	Die Plastkarte mit Magnetstreifen	12
1.2.1.4.1.	Aufbau der Karte	12
1.2.1.4.2.	Aufzeichnung der Daten	12
1.2.1.5.	Behandlung der Karten	13
1.2.2.	Anwenderschnittstelle	13
1.2.2.1.	Allgemeines	13
1.2.2.2.	Aufrufsequenz des Treibers	13
1.2.2.3	Wichtige Hinweise zur Funktionsauswahl	16
1.2.3.	Erlaeuterung der Funktionen	16
1.2.3.1.	Initialisierung	16
1.2.3.2.	Eingabe in Speicher (Lesen)	16
1.2.3.3.	Ausgabe auf Magnetkarte (Schreiben)	17
1.2.3.4.	Magnetkarte beschlagnahmen (MBE)	17
1.2.3.5.	Rueckwaerts in Schreib-Leseanfangsposition (RSL)	18
1.2.3.6.	Magnetkarte aus SLE ausstossen (MAS)	18
1.2.3.7.	Entnahmeaufforderung einschalten (EAE)	18
1.2.3.8.	Eingabetor der SLE verriegeln (ETV)	18
1.2.3.9.	Statustest	18
1.2.3.10.	Melden der Kartenposition (MKP)	19
1.3.	Treiber zum asynchronen seriellen Datenaustausch ueber die V.24-Schnittstelle der STE K 8025 oder K 6028 AP62 1520 (SCPX)	20
1.3.1.	Allgemeines	20
1.3.2.	Protokoll	21
1.3.3.	Aufruf	21
1.3.4.	Beendigung	24

		Seite
2.	Programmierempfehlungen und Demonstrationsprogramme zur BASIC-Anwendung	28
2.1.	Standardmoduln zur Unterstuetzung der Menuegestaltung	28
2.1.1.	Statisches Menue	28
2.1.2.	Rollen eines Teiles des Bildschirminhaltes	29
2.1.3.	Maskierte Eingabe	31
2.2.	Sortierprogramme	33
2.2.1.	Sortierprogramm 1	33
2.2.2.	Sortierprogramm 2	37
2.2.3.	Sortierprogramm 3	40
2.3.	Verarbeitung grosser Diskettendatenmengen	43
2.4.	Demonstrationsprogramme	50
2.4.1.	TREFFER	50
2.4.2.	KALENDER	59
2.4.3.	WOCHENTAG	63
3.	Zusatzpaket 1/2"-Magnetbandunterstuetzung im BASIC-Compiler BASICMB 1520 (SCPX)	73
3.1.	Technologie	73
3.1.1.	Compilieren BASIC-Programm	73
3.1.2.	Verbinden Objektprogramm	73
3.1.3.	Ueberlagerung mit MB-Steuerroutine	73
3.1.4.	Ausgabe BASIC-Programm auf Diskette	74
3.2.	Aufruf Magnetbandroutine in BASIC	74
3.3.	Fallbeispiel	75
3.4.	Dateien	75
4.	Ergaenzungen zum FORTRAN- und C-Compiler	81
4.1.	Ergaenzungen von FORTRAN-Compiler-Anweisungen	81
4.2.	Version 1.1 zur E/A-Bibliothek und Zusatzpaket 1/2"-MB-Unterstuetzung im C-Compiler	87
5.	Routine fuer schluesselindizierten Zugriff KISAMC 1520 (SCPX)	131
5.1.	Charakteristik und Leistungsumfang	131
5.2.	Anwendungsbedingungen	133
5.3.	Datei-Funktionen	135
5.3.1.	OPEN-Funktion	135
5.3.2.	CLOSE-Funktion	137
5.3.3.	KEY-Funktion	137
5.3.4.	READ-Funktion	139
5.3.5.	EXTEND-Funktion	140
5.3.6.	UPDATE-Funktion	141
5.4.	Uebersicht ueber die Rueckkehrcodes	142
5.5.	Arbeitsvarianten mit KISAMC	149
5.6.	Multvolume-Arbeit mit KISAMC	151

5.6.1. Arbeitsweise beim Neuanlegen Multivolumedatei 151
5.6.2. Arbeitsweise beim Erweitern Multivolumedatei 152
5.6.3. Arbeitsweise beim Lesen Multivolumedatei 152
5.6.4. Arbeitsweise mit schluesselindiziertem Zugriff auf eine Multivolumedatei 153
5.6.5. Arbeitsweise mit UPDATE in einer Multivolumedatei 154

1. Treiber-Unterprogramme

1.1. Treiber fuer Kassettenmagnetbandgeraet K 5200 KMBG 1520 (SCPX)

1.1.1. Aufgabe des Programms

Das Unterprogramm realisiert Basisfunktionen der Eingabe und Ausgabe von Datenbloecken sowie Steuerfunktionen des Kassettenmagnetbandgeraetes K 5200.00. Das K 5200.00 ist ueber die Anschlusssteuerung AKB K 5020 zu betreiben. Das physische Aufzeichnungsverfahren ist nach KROS 5109 gewaehlt. Die Basisfunktionen gestatten die Arbeit im "Simple"-Niveau. Die darauf aufbauenden Niveaus, einschliesslich Dateibearbeitungsfunktionen, sind vom Anwender selbst zu organisieren. Es koennen wahlweise zwei KMBG angesteuert werden.
Um eine hohe Datensicherheit zu erreichen, werden folgende Kontrollen realisiert:

- Die Aufzeichnung erfolgt im read-after-write-Verfahren mit anschliessendem direkten Speichervergleich (1:1- Vergleich). Damit wird eine maximale Sicherheit gewaehrleistet.

- Leseoperationen werden mit CRC-Kontrolle durchgefuehrt.

1.1.2. Aufrufsequenz des KMBG-Treibers

- Der Aufruf der einzelnen Funktionen erfolgt mittels Parameteruebergabe an die CPU-Register und anschliessendem Unterprogrammaufruf

  ```
  CALL KMBG.
  ```

  Die Funktionsaufrufe werden in dieser Schrift im U880-Assembler-Niveau dargestellt.

- Vom Anwender sind folgende Uebergabeparameter bereitzustellen:

Register	Bedeutung
C	Funktionscode (FC)
HL	Blocklaenge fuer Lesen/Schreiben 1 Block HL=2-100H (Blocklaenge kann 2 bis 256 Datenbytes betragen)
DE	Anfangsadresse des Datenbereiches fuer Lesen/Schreiben 1 Block

- Uebersicht der Funktionscodes:

NR	Funktion	FC fuer KMB1	FC fuer KMB2
0	- Lesen 1 Block	01H	41H
1	- Schreiben 1 Block	02H	42H
2	- Schreiben 1 Bandmarke	11H	51H
3	- Vorsetzen 1 Block	12H	52h
4	- Ruecksetzen 1 Block	32H	72H
5	- Vorsetzen 1 Bandmarke	13H	53H
6	- Ruecksetzen 1 Bandmarke	33H	73H
7	- Schnelles Vorspulen	14H	54H
8	- Schnelles Rueckspulen	34H	74H
9	- Verriegeln/Reservieren	15H	55H
10	- Entriegeln	16H	56H
11	- Band loeschen	18H	58H

Rueckmeldungen des Treibers ueber CPU-Register:

- A = Statusbyte
  HL = tatsaechlich gelesene physische Zeichenanzahl eines Blocks (Anzahl Datenbytes + 4 Bytes)

- Struktur des Statusbytes (SB):

bit 0 = 1: falscher Funktionscode bzw. Ueberschreitung der zulaessigen Zeichenzahl eines Blocks

bit 1 = 1: Blocklaengenfehler, d.h., programmierte Zeichenzahl stimmt nicht mit gelesener Zeichenzahl (Rueckmeldung in HL) ueberein. Es kann eine Unterschreitung bzw. Ueberschreitung vorliegen
oder:
Soforterkennung Bandmarke (bei FC = 13H/53H bzw. 33H/73H), siehe 1.1.4.5.

bit 2 = 1: Geraet nicht bereit oder:
Zeitlimitfehler, d.h. angewaehlte Funktion konnte innerhalb eines Zeitlimits von ca. 1s nicht ausgefuehrt werden. Die Ausfuehrung aller Bandoperationen wird ueberwacht. Der Zeitlimitfehler wird ausgegeben, wenn innerhalb 1s kein Flusswechsel auf dem Band erkannt wird. Bei Bandstillstand ist das der Fall. Dies kann auftreten bei Hardwarefehlern und bei fehlerhafter Programmbedienung des Bandes (z.B. Bandanfangs- oder Bandendeklarsichtband ist erreicht).

bit 3 = 0: nicht genutzt, d.h. immer "0"

bit 4 = 1: R/W- Fehler

bit 5 = 1: BM erkannt

bit 6 = 1: Klarsichtband

bit 7 = 0: nicht genutzt, d.h. immer "0"

1.1.3. Programmbedingungen

- Das Unterprogramm belegt ca. 3450 Bytes.
- Fuer die Ansteuerung zweier KMBG K5200 mittels Adaptersteckeinheit AKB K5020 werden die E/A-Tore 30H...37H belegt. Zusaetzlich wird Kanal 1 des CTC-Schaltkreises auf der ZRE benoetigt (EA-Tor: 0DH).
- Der Treiber arbeitet im Interruptbetrieb.
- In der durch die einzelnen Treiber dynamisch verwalteten Interrupt-Saeule werden 8 Bytes benoetigt.

1.1.4. Erlaeuterung der Funktionen

Vor Anwahl einer KMBG-Funktion ist als erstes das ausgewaehlte Laufwerk ueber die Funktion " Reservieren" zu verriegeln. Analog muss nach Beendigung der Kassettenarbeit das jeweilige Laufwerk wieder entriegelt werden, da sonst die Kassette nicht entnommen werden kann.
Fuer alle Funktionen 1.1.4.1. bis 1.1.4.5. gilt, dass bei Ersterkennung von Klarsichtband im SB Bit 6 = 1 gesetzt wird.

1.1.4.1. Lesen 1 Block

Es wird der naechste Block in Vorwaertsrichtung gelesen. Stimmt die Anzahl der Datenbytes des gelesenen Blocks nicht mit der programmierten Zeichenzahl ueberein, wird im SB das Bit 1 gesetzt.
In HL wird die Anzahl der tatsaechlich gelesenen Bytes angegeben, d.h. Anzahl Datenbytes des Blockes, je ein Byte Postambel und Praeambel sowie zwei Bytes CRC-Zeichen. Im E/A-Bereich wird nur die programmierte Anzahl von Datenbytes abgelegt. Erkennt beim Lesen die CRC-Kontrolle einen Fehler, erfolgt Lesewiederholung. Erst nach fuenfmaligem Lesen mit negativer CRC-Kontrolle erfolgt die Meldung durch Bit 4 = 1 im SB.
Wird beim Lesen eine BM erkannt, wird Bit 5 im SB gesetzt.

1.1.4.2. Schreiben 1 Block

Es wird 1 Block mit der in HL angegebenen Blocklaenge aufgezeichnet. Dabei werden durch den Treiber Praeambel, Postambel sowie zwei CRC-Bytes dem Block angefuegt.
Aufzeichnungsfehler werden durch das read-after-write-Verfahren und anschliessenden Speichervergleich innerhalb des gesamten Blockes festgestellt. Im Fehlerfall wird der Schreibvor-

gang bis zu 8 mal wiederholt, erst danach erfolgt die entsprechende Fehlermeldung im SB (Bit 4 = 1).

1.1.4.3. Schreiben Bandmarke

Es wird eine BM, bestehend aus 4 Bytes (Praeambel, 1.NUL-Byte, 2.NUL-Byte, Postambel), aufgezeichnet.

1.1.4.4. Vor-/Ruecksetzen 1 Block

Das Magnetband wird um 1 Block vor- bzw. zurueckgesetzt. War dieser Block eine BM, wird Bit 5 = 1 im SB gesetzt.

1.1.4.5. Vor-/Ruecksetzen 1 Bandmarke

Das Magnetband wird solange vor- bzw. rueckgesetzt, bis eine BM erkannt wird. Das wird durch Setzen von Bit 5 = 1 im SB dem Anwender mitgeteilt. Ist der erste Block ueber den vor- bzw. rueckgesetzt wird eine BM, wird das zusaetzlich durch Setzen von Bit 1 = 1 im SB kenntlich gemacht (Soforterkennung BM).

1.1.4.6. Schnelles Vor- bzw. Rueckspulen

Das Band wird im Schnellauf ($1{,}5 ms^{-1}$) bis zum Klarsichtband vor- bzw. rueckgespult.
Dieser Vorgang laeuft bis zum Bandstop unter ZRE-Steuerung ab, d.h., die Kassette spult nicht autonom zurueck.

1.1.4.7. Ver- bzw. Entriegeln KMB

Die Funktion "Verriegeln KMB" ist die Voraussetzung fuer alle bisher aufgezaehlten Funktionen. Wird auf das KMB im nicht verriegelten Zustand zugegriffen, erfolgt die Meldung "Geraet nicht bereit" durch Bit 2 = 1 im SB.

Die Funktion "Entriegeln KMB" bedeutet fuer den Anwender, dass die Arretierung der Entladetaste aufgehoben wird und dadurch die Kassette entnommen werden kann.

1.1.4.8. Band loeschen

Mit dieser Funktion wird ab aktueller Bandposition das Band bis zum Klarsichtband nach EOT geloescht.

1.2. Treiber fuer den Anschluss der Schreib-Leseeinheit SLE 1520 (SCPX)

1.2.1. Hardware

1.2.1.1. Schreibleseeinheit und Anschlusssteuerung

Die Schreibleseeinheit (SLE) K 6501 ist ein Geraet zum Lesen und Beschreiben von Plastkarten mit Magnetstreifen und an den BC A5120/30 bzw. an Terminals anschliessbar.
Der Anschluss erfolgt mit der Anschlusssteuerung K 6001 fuer eine SLE. Die Anschlusslaenge der SLE an die K 6001 betraegt maximal 5m.
Die Anschlusssteuerung K 6001 dient der SLE als Kopplung an die ZRE K 2526 ueber das Linieninterface BUS K 1520.
Sie fuehrt den Austausch der Daten und der erforderlichen Steuersignale fuer das Lesen und Beschreiben der Karten in der SLE durch.
Ueber einen 26-poligen Steckverbinder ist eine SLE anschliessbar.

1.2.1.2. Technische Daten

- **Bestimmungsangaben**

Datentraeger	:	Plastkarte mit Magnetstreifen nach TGL 42091, 42092, 42093, ISO 3554, DIN 9781,9785
Datenspuren	:	3
Aufzeichnungsverfahren	:	Wechseltaktschrift nach DIN 66010
Aufzeichnungscode nach ISO 3554	:	Spur1: 6-Bit-Code Spur2: 4-Bit-Code Spur3: 4-Bit-Code
Schreibkapazitaet	:	Spur1: 76 nutzbare Zeichen Spur2: 37 nutzbare Zeichen Spur3: 104 nutzbare Zeichen und 3 Sonderzeichen pro Spur
Spurdichte	:	Spur1: 8,3 Bit/mm +/-5% Spur2: 3,0 Bit/mm +/-3% Spur3: 8,3 Bit/mm +/-8%
Kartengeschwindigkeit	:	80mm/s
Magnetkopf	:	3-Spur-Aufzeichnungs-/Wiedergabekopf

Anschlussbedingungen am K 2526	:	BUS 1520/ASL K 6001
Anschlussverbindung	:	Stecker 26-polig, TGL 16561
Anschlusslaenge	:	maximal 5m

- Gewicht und Abmessungen

Abmessungen	:	Hoehe 115mm Breite 135mm Tiefe 270mm
Gewicht	:	2,4kg

- Sondervariante der SLE

SLE mit Sicherheitsvorsatz und Beschlagnahmeeinrichtung (nur fuer Sonderanwender verfuegbar).

- Leistungsaufnahme

Stromversorgung	:	extern
Gesamt-Leistungs-aufnahme	:	maximal 4,5W
Betriebsspannungen	:	+ 5V +/- 5% +12V +/- 5% -12V +/- 5%

Die SLE wird als Beistellgruppe (ohne eigene Stromversorgung) ueber ein Kabel (Zubehoer) an die Anschlusssteuerung angeschlossen

1.2.1.3. Fehlererkennung

Es erfolgt bei der SLE die Kontrolle der Karten durch die Anschlusssteuerung K 6001. Jedes Datenzeichen ist durch ein Paritaetsbit auf ungerade Paritaet ergaenzt.
Der Datensatz (Datenzeichen einer Spur) wird durch ein LRC-Zeichen abgeschlossen, dass alle Datenbits gleicher Wertigkeit (ausser dem Paritaetsbit) auf gerade Paritaet ergaenzt. Das LRC-Zeichen selbst wird auf ungerade Paritaet ergaenzt.

Bei Verwendung der speziellen SLE Variante mit Sicherheitsvorsatz erfolgt eine Kontrolle der Karten durch die Hardware. Hier erfolgt eine Eingabesperre bei Verwendung von nicht normgerechten Karten oder beim Einstecken von falschen Karten in die SLE.

1.2.1.4. Die Plastkarte mit Magnetstreifen

1.2.1.4.1. Aufbau der Karte

Als Datentraeger wird die Plastkarte mit Magnetstreifen nach ISO 3554 verwendet.
Die Karte besteht aus Kunststoff oder kunststofflaminiertem Werkstoff und ist einseitig mit einem Magnetstreifen versehen.

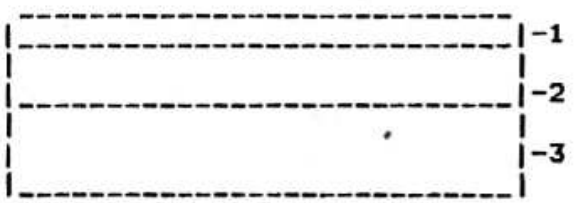


Bild 1: Aufbau der Karte

1.2.1.4.2. Aufzeichnung der Daten

Auf dem Magnetstreifen erfolgt die Datenspeicherung in 3 parallelen Spuren.

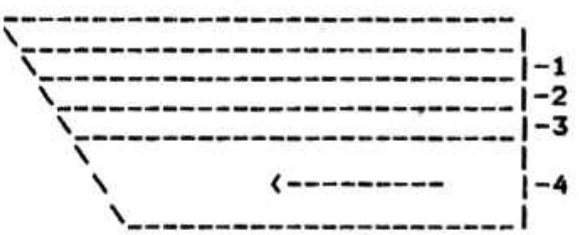


Bild 2: Spuraufbau der Karte

Die Aufzeichnung eines Zeichens beginnt mit dem niederwertigsten Bit und endet mit dem Paritaetsbit.

Fuer die Spuren 2 und 3 erfolgt die Datenaufzeichnung im 4-Bit-Code und fuer die Spur 1 im 6-Bit-Code nach ISO 3554.

Die Spuren haben folgenden physischen Aufbau:

Spur 1 : 1 Startzeichen
76 alphanumerische Zeichen (nutzbare Zeichen fuer Anwender)
1 Stopzeichen
1 Pruefzeichen (LRC-Zeichen)

Spur 2 : 1 Startzeichen
37 numerische Zeichen (nutzbare Zeichen fuer Anwender)
1 Stopzeichen
1 Pruefzeichen (LRC-Zeichen)

Spur 3 : 1 Startzeichen
104 numerische Zeichen (nutzbare Zeichen fuer Anwender)
1 Stopzeichen
1 Pruefzeichen (LRC-Zeichen)

1.2.1.5. Behandlung der Karten

Sichtbare Verschmutzungen der Kartenoberflaeche sind vor dem Eingeben in die SLE zu entfernen.
Verletzungen der Magnetstreifenoberflaeche sind zu vermeiden.
Zum Schutz vor Verschmutzung und mechanischer Beschaedigung sind die Magnetsreifen mit einer mehrere Mikrometer dicken Deckschicht aus durchsichtigem, nicht magnetisierbarem Material versehen.
Die Karten sind vor Fremdmagnetfeldern zu schuetzen.

1.2.2. Anwenderschnittstelle

1.2.2.1. Allgemeines

Der Treiber wird als REL-Datei bereitgestellt (verschiebliche Objektcodedatei).

Der Treibername ist SLE. Dieses Symbol muss der Anwender im aufrufenden Modul als externes Symbol definieren.

Der Treiberaufruf erfolgt mittels

```
CALL SLE
```

Die Parameteruebergabe an den Modul erfolgt ueber die CPU-Register. Diese sind vor Aufruf zu laden. Register werden im Treiber nicht gerettet. Dies ist bei Bedarf vor Aufruf des Treibers vom aufrufenden Modul zu realisieren.
Der Treiber besitzt einen Stackbedarf von 32 Byte. Diesen Bereich muss der aufrufende Modul mindestens zur Verfuegung stellen.
Der Treiber arbeitet im Interruptbetrieb (INT-MODE 2) und belegt innerhalb der Interruptsaeule (0F740H - 0F7DFH) 16 Bytes.

1.2.2.2. Funktionsauswahl

Der Aufruf der einzelnen Funktionen erfolgt mittels Parameteruebergabe an die CPU-Register und anschliessendem Unterprogrammaufruf:

```
Register  C - Funktionscode
Register DE - Speicheradresse fuer E/A
CALL          SLE
```

- Uebersicht der Funktionscodes

Register	/Nr.der Funktion und Wirkung
C = 00	/0/ SLE in Grundstellung (Initialisierung)
C = 11 DE= adr.	/1/ Eingabe in Speicher(Lesen) Spur 1 Speicheradresse
C = 12 DE= adr.	/2/ Eingabe in Speicher(Lesen) Spur 2 Speicheradresse
C = 13 DE= adr.	/3/ Eingabe in Speicher(Lesen) Spur 3 Speicheradresse
C = 14 DE= adr.	/4/ Eingabe in Speicher(Lesen) Spuren 1/2 Speicheradresse
C = 15 DE= adr.	/5/ Eingabe in Speicher(Lesen) Spuren 2/3 Speicheradresse
C = 21 DE= adr.	/6/ Ausgabe auf Magnetkarte (Schreiben) Spur 1 Speicheradresse
C = 22 DE= adr.	/7/ Ausgabe auf Magnetkarte (Schreiben) Spur 2 Speicheradresse
C = 23 DE= adr.	/8/ Ausgabe auf Magnetkarte (Schreiben) Spur 3 Speicheradresse
C = 24 DE= adr.	/9/ Ausgabe auf Magnetkarte (Schreiben) Spuren 1 oder 2; Speicheradresse
C = 25 DE= adr.	/10/ Ausgabe auf Magnetkarte (Schreiben) Spuren 2 oder 3; Speicheradresse
C = 30	/11/ Magnetkarte beschlagnahmen (MBE) *)
C = 40	/12/ Rueckwaerts in Schreib-Leseanfangsposition (RSL) *)
C = 50	/13/ Magnetkarte aus SLE ausstossen (MAS) *)
C = 60	/14/ Entnahmeaufforderung einschalten (EAE) *)
C = 70	/15/ Eingabetor der SLE verriegeln (ETV) *)
C = 80	/16/ Statustest *)
C = 90	/17/ Melden der Kartenposition (MKP) *)

*) DE-Registerinhalt fuer die Funktion bedeutungslos und kann beliebig eingestellt sein

- **Statusmeldungen**

In der Funktion 16 "Statustest" liefert der Treiber im Register A dem aufrufenden Modul folgende Rueckmeldungen:

Register	Bedeutung
A=FF	SLE nicht initialisiert bzw. INT-Tabelle nicht geladen
A=FE	Lesen initialisiert, aber noch keine Karte in SLE
A=FC	Schreiben initialisiert, aber noch keine Karte in SLE
A=7F	Lesen/Schreiben /MBE/RSL/MAS, aber mit Fehler beendet
A=00	Lesen/Schreiben /MBE/RSL/MAS ohne Fehler beendet; korrekt

Funktionen 1-10:
- Funktionen mittels eines programmierten Funktionsaufruf 16 beenden
- "Statustest" beendet Funktion, wenn Karte in SLE eingesteckt wird, sonst nicht

Funktionen 11, 12, 13:
- Funktionen werden sofort ausgefuehrt und liefern selbst die Statusmeldung. Die Statusrueckmeldung kann im schlechtesten Fall erst 5s nach Funktionsaufruf erfolgen (Kartendurchlaufzeit).

Funktionen 14, 15:
- Funktionen arbeiten ohne Kartentransport und geben keine Statusmeldung zurueck

- **Positionsmeldungen**

Die Funktion 17 (MKP) traegt die Kartenposition in der SLE im Register A ein:

Register	Kartenposition
A =00	keine Karte im Leser
A =01	Karte in Einzugsposition
A =02	Karte in Schreib-Lese-Anfangsposition
A =04	Karte in Schreib-Lese-Endposition

1.2.2.3. Wichtige Hinweise zur Funktionsauswahl

Nach dem Aufruf der Funktion 11 (MBE) duerfen folgende Funktionen nicht programmiert werden:

Funktionen	6,7,8,9,10	Schreiben
Funktion	11	MBE
Funktion	12	RSL
Funktion	13	MAS

1.2.3. Erlaeuterung der Funktionen

1.2.3.1. Initialisierung

Mit der Funktion 0 wird die SLE in Grundstellung gebracht. Das ist vor dem ersten Funktionsaufruf notwendig. Mit der Funktion 0 kann ebenfalls ein sofortiger Abbruch einer laufenden Funktion erfolgen.

1.2.3.2. Eingabe in Speicher (Lesen)

Die Funktionen 1, 2, 3, 4, 5 bewirken die fortlaufende Uebernahme der Daten in den Speicher ab der im Register DE angegebenen Adresse.
Werden 2 Spuren gleichzeitig gelesen, stehen die Daten in folgender Reihenfolge im Speicher:

```
SPUR 1 , SPUR 2     bei Funktion 4
SPUR 2 , SPUR 3     bei Funktion 5
```

Beispiel:

```
C=14   DE=adr
Lesen der Karte erfolgt in der Reihenfolge Spur 1,
Spur 2.
Ab adr sind 113 Bytes vom aufrufenden Modul zu reservieren (Spur 1 = 76 Bytes, Spur 2 = 37 Bytes).
```

Die 4- bzw. 6-Bit-Daten von der Karte werden vom Treiber in den ISO-7-Bit-Code gewandelt.
Start-, Stopp- und LRC-Zeichen werden nicht in den Speicher eingelesen. Trennzeichen und Sonderzeichen innerhalb der Daten werden als Daten eingelesen. Am Ende der Leseoperation befindet sich die Karte an der Leseendposition.

Mit der Funktion 16 "Statustest" muss der aufrufende Modul anschliessend die Richtigkeit der ausgefuehrten Funktion "Lesen" ueberpruefen. Danach kann die Karte mit der Funktion 13 (MAS) ausgetrieben werden. Eine Lesewiederholung bei Fehlern findet nicht statt. Nach Lesefehlern kann mit der Funktion 12 (RSL) die Karte in Leseanfangsposition rueckpositio-

niert werden. Mit erneutem Funktionsaufruf "Lesen" kann der Lesevorgang wiederholt werden.
Die Funktion "Lesen" wird abgebrochen, wenn auf dem Magnetstreifen kein gueltiges Startzeichen erkannt wird. Die Karte befindet sich in der Leseendposition. Es wurden keine Daten in den Speicher eingelesen. Der Speicherinhalt ab adr ist als "undefiniert" zu betrachten. Es erfolgt eine Fehlermeldung ueber das Register A.

Es wird ein festes Laengenformat entsprechend der Spurkapazitaet (Spur 1: 76 Bytes, Spur 2: 37 Bytes, Spur 3: 104 Bytes) gelesen.

1.2.3.3. Ausgabe auf Magnetkarte (Schreiben)

Die Funktionen 6, 7, 8, 9, 10 bewirken das Aufzeichnen der im Speicher im ISO-7-Bit-Code ab adr befindlichen Daten auf die Karte.
Sind 2 Spuren gleichzeitig zu beschreiben, werden die Daten in der Reihenfolge

SPUR 1 , SPUR 2	bei Funktion 9
SPUR 2 , SPUR 3	bei Funktion 10

aufgezeichnet.

Die Aufzeichnung erfolgt (im 4- oder 6-Bit-Code) auf die gewuenschte Spur. Start-, Stopp- und LRC-Zeichen werden vom Treiber erzeugt. Sie sind nicht im Ausgabebereich bereitzustellen. Trenn- und Sonderzeichen muessen wie die Daten im Ausgabebereich bereitgestellt sein.

Nach dem Schreiben befindet sich die Karte in der Schreibendposition. Es findet kein Kontolllesen statt.

Analog der Funktion "Lesen" muss der aufrufende Modul nach "Schreiben" mittels "Statustest" ueberpruefen, ob die Ausgabe fehlerfrei ausgefuehrt wurde. Liegt kein Fehler vor, kann die Karte mittels Funktion 13 (MAS) ausgetrieben werden.

Im Fehlerfall findet keine Schreibwiederholung statt. Nach Rueckpositionierung der Karte (Funktion 12 (RSL)) kann der Schreibvorgang wiederholt werden.

1.2.3.4. Magnetkarte beschlagnahmen (MBE)

Die Beschlagnahme der Karte ist technisch nur bei speziellen Ausfuehrungen der SLE moeglich. Es muss ein Ablagefach fuer die beschlagnahmten Karten vorhanden sein .

Wird die Funktion 11 (MBE) aufgerufen, bleibt die Karte vorerst in der Kartenbahn des Geraetes liegen. Sie wird nicht ausgetrieben.

Die naechste Karte, die gelesen bzw. beschrieben wird, schiebt die vorhergehende Karte ins Ablagefach.

1.2.3.5. Rueckwaerts in Schreib-Leseanfangsposition (RSL)

Es findet ein Rueckpositionieren der Karte aus der Schreib- bzw. Leseendposition in die Schreib- bzw. Leseanfangsposition statt. Danach kann ein erneuter Schreib- bzw. Lesebefehl erfolgen.
Die Karte gelangt nicht in den Zugriffsbereich des Anwenders.

1.2.3.6. Magnetkarte aus SLE ausstossen (MAS)

Die Karte wird aus der SLE ausgetrieben und gelangt in den Zugriffsbereich des Anwenders. Sie kann entnommen werden.

1.2.3.7. Entnahmeaufforderung einschalten (EAE)

Am Anschlusspunkt X2/A8 des Koppelbusssteckers auf der Anschlusssteuerung K 6001 (siehe Betriebsdokumentation K 6001) wird ein Rechteckimpuls ausgegeben. Der Nutzer kann an diesen Punkt eine anwendereigene Elektronik anschliessen.

Nutzungsbeispiel:

> Wurde eine Karte nach einer festgelegten Zeit nach Beendigung eines Schreib- bzw. Lesevorganges nicht aus der SLE entnommen, kann der Anwender die Funktion 14 (EAE) aufrufen. Daraufhin wird dieses Signal zur Ansteuerung einer Entnahmeaufforderung (Lampe oder akustisches Signalgeber) genutzt.

1.2.3.8. Eingabetor der SLE verriegeln (ETV)

Diese Funktion des Treibers ist nur an der SLE mit Sicherheitsvorsatz nutztbar. Der Eingabesektor wird verriegelt, d.h. es erfolgt eine Eingabesperre. Der Karteneinzug ist folglich gesperrt.

1.2.3.9. Statustest

Mit der Funktion 16 "Statustest" ueberprueft der Treiber das Ergebnis der vorausgegangenen Operation. Der Status wird (siehe Punkt 1.2.2.2.) im Register A bereitgestellt.

Die Funktionen "Eingabe in den Speicher" (Lesen) und "Ausgabe auf Magnetkarte" (Schreiben) sind stets mit dem "Statustest" zu kontrollieren.
Es erfolgt die Pruefung auf die physische Beendigung der Funktion "Lesen". Wird dies festgestellt, prueft die LRC-Kontrolle die gelesenen Daten.

Im Fehlerfall wird im Register A der Fehlercode bereitgestellt.
Stellt der "Statustest" die Beendigung des Lesens bzw. Schreibens fest, wird der Grundzustand (Zustand nach dem Initialisieren) hergestellt. Dies ist auch bei LRC-Fehler der Fall.

Die Funktionen 11, 12, 13 melden ihren Status selbst im Register A und bringen den Treiber in den Grundzustand.

1.2.3.10. Melden der Kartenposition (MKP)

Die Kartenposition innerhalb der SLE wird im Register A gemeldet (siehe Positionsmeldungen).

1.3. Treiber zum asynchronen seriellen Datenaustausch ueber die V.24-Schnittstelle der STE K 8025 oder K 6028 AP62 1520 (SCPX)

1.3.1. Allgemeines

Das Treiberunterprogramm AP62 ermoeglicht nach Protokoll analog AP 62/64 den asynchronen seriellen Datenaustausch ueber die Schnittstelle V.24 der Steckeinheit K 8025 oder K 6028 des BC A5120 und A5130.
AP62 wird als verschiebliches Unterprogramm (REL-Datei) zum Einbinden in Anwenderprogramme bereitgestellt. Fuer den Unterprogrammaufruf und die Parameteruebergabe muss im rufenden Programm

- die Definition externer Symbole,
- die Parameteruebergabe in Register HL und
- der Unterprogrammaufruf

programmiert werden.

Das Senden und Empfangen erfolgt interruptgesteuert. Dazu werden 8 Bytes in der Interrupt-Saeule des SCPX belegt. Zur Erzeugung des Sende- und Empfangsschrittaktes und zur Realisierung der Zeitablaufsteuerung werden 2 Kanaele der CTC der ZRE genutzt (Toradressen OCH und OFH).
Auf der STE K 8025 oder K 6028 werden die SIO-Kanaele mit den Toradressen 50H, 51H und 53H genutzt.
Der DIL-Schalter A45 auf der STE K 6028 bzw. A46 auf der STE K 8025 muss wie folgt eingestellt werden:

Schalter-Anschluesse K 6028	K 8025	Stellung
1 - 16	1 - 8	AUS
2 - 15	2 - 7	EIN
3 - 14	3 - 6	EIN
4 - 13	4 - 5	AUS

Das Treiberunterprogramm laesst zwei Betriebsarten des BC zu:

a) Arbeit als Hauptstation (master)
b) Arbeit als Nebenstation (slave)

Bei der Arbeit als Hauptstation kann der BC Kommandofolgen ausgeben und damit die Nebenstationen steuern.
Bei Arbeit als Nebenstation erfolgt eine Reaktion des BC nur bei korrektem Empfang einer Kommandofolge mit der eigenen Stationsadresse.
Zur Anpassung an den Fernmeldeweg gelangen Modems bzw. GDN's zum Einsatz. Unterstuetzt wird durch den Treiber die Betriebsart

asynchron/halbduplex/Uebertragungsgeschwindigkeit $\leq$ 9600 bit/s

der Datenuebertragungseinrichtungen.

1.3.2. Protokoll

Das Protokoll analog AP62/64 ist in der Dokumentation "Anleitung fuer den Systemprogrammierer fuer Geraete mit dem Betriebssystem SIOS 1526, Teil Datenfernuebertragung" S.24 ff beschrieben.

1.3.3. Aufruf

Das Treiberunterprogramm besitzt die 5 Eintrittspunkte:

OP, RD, WR, WK, CL.

Der Aufruf (Ruf) erfolgt mit einem CALL an eines der 5 globalen Symbole (CALL OP, CALL RD, CALL WR, CALL WK, CALL CL). Dabei muss im Register HL die Adresse eines Parameterfeldes uebergeben werden. Es gilt generell, dass die Parameter nicht auf ihre Zulaessigkeit geprueft werden.

- Ruf OP

- Bedeutung: Der Ruf dient dem logischen Ankoppeln an den Fernmeldeweg. Mit diesem Ruf werden festgelegt bzw. ausgewaehlt :
  - Haupt- oder Nebenstation
  - Stationsadresse
  - Uebertragungsgeschwindigkeit
  - Groesse des Empfangspuffers
  - Blocklaenge
  - Adresse des Unterbrechungsprogramms (siehe auch 1.3.4.)

- Aufbau des Parameterfeldes:

Parameter	Laenge (Byte)	Bedeutung
ADR1	1	Stationsadresse; Angabe als ISO-Zeichen
MASL	1	Auswahl Haupt-/Nebenstation H'00' = Nebenstation H'02' = Hauptstation
BL	2	Blocklaenge; Angabe als binaerer Wert
LRB	2	Groesse des Emfangspuffers (wird diese ueberschritten, so wird Ueberlauf gesetzt); Angabe als binaerer Wert

Parameter	Laenge (Byte)	Bedeutung
USINT	2	Anfangsadresse einer Assembler-Routine, zu der nach beendeten Rufen RD, WR oder WK verzweigt wird

Es entsteht ein Off-line-Fehler, wenn das Modem nicht nach spaetestens 0,3s nach Einschalten der Leitung V.108 ("Uebertragungsleitung anschalten") durch den BC die Leitung V.107 ("Betriebsbereitschaft") zurueckmeldet. Dabei wird automatisch ein Ruf CL abgearbeitet und auf den Bildschirm die Meldung "AP 62/64 OFF-LINE" ausgegeben.

- Ruf RD

- Bedeutung: Dieser Ruf sendet eine Sendeaufforderung (Polling) an die Nebenstation und empfaengt daraufhin eine oder mehrere Nachrichten. Das Senden einer Textantwort ist wahlweise moeglich.

- Aufbau des Parameterfeldes:

Parameter	Laenge (Byte)	Bedeutung
BAR	2	Empfangspufferadresse
BAT	2	Adresse des Sendepuffers; wird Adresse 0 angegeben, wird ohne Textantwort gearbeitet
MLT	2	Sendelaenge = Anzahl der zu sendenden Zeichen bei Textantwort; Angabe als binaerer Wert
ADR 1 ADR 2 KDO	1 1 1	Adress- bzw. Kommandozeichen der zu sendenden Kommandofolge Angabe als ISO-Zeichen

- Ruf_WR

- Bedeutung: Dieser Ruf sendet eine Empfangsaufforderung (Selecting) an die Nebenstation und sendet bei positiver Quittung den entsprechenden Text.

- Aufbau des Parameterfeldes:

Parameter	Laenge (Byte)	Bedeutung
BAT	2	Adresse des Sendepuffers
MLT	2	aktuelle Sendelaenge
ADR 1	1	) Adress- bzw. Kommandozeichen
ADR 2	1	) der Selecting-Folge;
KDO	1	) Angabe als ISO-Zeichen

- Ruf_WK

- Bedeutung: Dieser Ruf empfaengt eine Kommandofolge und speichert die Bytes ADR 2 (Komponentenadresse) und KDO (Kommando) auf den Adressen 000EH und 000FH ab. Ist zum Zeitpunkt des Eintreffens einer Kommandofolge kein "WK" aktiv, wird die Assemblerroutine, deren Adresse unter USINT beim Ruf OP angegeben wurde, mit dem Status "Kommandofolge empfangen" angesprungen.
Der gleiche Ablauf ergibt sich, wenn zwar ein Ruf "WK" aktiv ist, aber die im Ruf angegebene Pufferadresse Empfang/Senden gleich 0 ist.
Ist nach spaetestens 2s kein Ruf "WK" aktiviert bzw. wird eine Pufferadresse = 0 definiert, wird automatisch EOT gesendet. Sonst erfolgt die Weiterarbeit entsprechend dem empfangenen Kommando. D.h., es wird Text entsprechend der Parameterfestlegung gesendet oder empfangen. Soll nach dem Senden von Text eine etwaige Textantwort der Hauptstation empfangen werden, ist eine Empfangspufferadresse ≠ 0 anzugeben. Ist die Empfangspufferadresse = 0, wird eine eintreffende Textantwort zurueckgewiesen und der Ruf mit dem Status "Pufferueberlauf" beendet.
Allgemein gilt, dass ein Ruf durch das Senden bzw. Empfangen von EOT oder durch Ueberschreiten des Quittungs-Time outs beendet wird.

- Aufbau des Parameterfeldes:

Parameter	Laenge (Byte)	Bedeutung
BAR	2	Adresse des Empfangspuffers
BAT	2	Adresse des Sendepuffers
MLT	2	Sendelaenge = Anzahl der zu sendenden Zeichen; Angabe als binaerer Wert

1.3.4. Beendigung

Nach jedem beendeten Ruf RD, WR oder WK wird zur Adresse verzweigt, die unter USINT im Ruf OP angegeben wurde. Das an dieser Adresse beginnende Unterprogramm wirkt aehnlich wie eine Interruptroutine. Deshalb sind alle verwendeten Register zu retten und ein eigener Stack anzulegen. Vor dem Beenden der Routine mit RET ist ausserdem auf Adresse 0009H das Bit 0 zurueckzusetzen und dem Treiberunterpogramm damit mitzuteilen, dass die Unterbrechung beendet ist. Die Auswertung des beendeten Rufes ist mit Hife des Steuerblocks moeglich. Dieser beginnt auf der festen Adresse 0008H und hat eine Laenge von 12 Bytes.

- Steuerblock:

Adresse	Inhalt	Bedeutung
0008H	SEB	Status-Error-Byte : Aus diesem Byte ist ersichtlich wie das letzte Kommando abgearbeitet wurde.
0009H	CB	Steuerbyte fuer Treiberunterprogramm : - Bit 0 = 1 Unterbrechung aktiv - Bit 2 = 0 Jeder Ruf wird beendet mit dem Senden von EOT bei BCT = 0 bzw. beim Empfang von EOT, d.h. wenn die Uebertragung beendet worden ist. - Bit 2 = 1 Jeder Ruf wird beendet mit dem Empfang der positiven Quittung (beim Senden) bzw. dem korrekten Empfang des letzten Blocks der Nachricht.
000AH-000BH	ACR	Adresszaehler Empfang : Gibt waehrend des Empfangs die Adresse an, auf der das naechste Empfangszeichen abgespeichert wird.

Adresse	Inhalt	Bedeutung
000CH-000DH	BCR	Bytezaehler Empfang : Gibt die Anzahl der korrekt empfangenen Zeichen an.
000EH	ADR2	Nach korrektem Empfang einer Kommandofolge wird hier das Zeichen ADR2 der Kommandofolge abgespeichert.
000FH	KDO	Nach korrektem Empfang einer Kommandofolge wird hier das Zeichen KDO der Kommandofolge abgespeichert.
0010H-0011H	ACT	Adresszaehler Senden : Gibt waehrend des Sendens die Adresse an, von der das naechste Zeichen ausgegeben wird.
0012H-0013H	BCT	Bytezaehler Senden : Restbytezaehler ! Gibt die Anzahl der noch zu sendenden (!) Zeichen an; Bei fehlerfreier Abarbeitung gilt BCT = 0.

Zur Fehlerdiagnose ist das SEB verwendbar. Dabei haben die Bits 0, 3, 6 und 7 allgemeine Bedeutung:

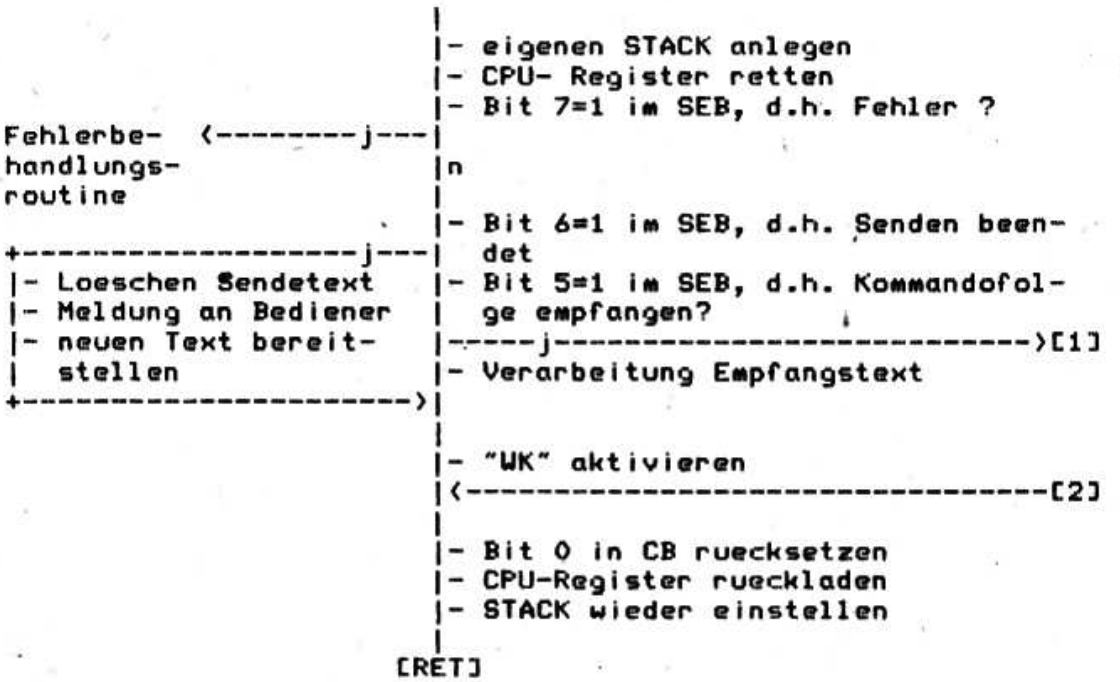
- Bit 0 = 1 Ueberlauf Empfangspuffer
- Bit 3 = 1 Hardwarefehler
- Bit 6 = 0 Empfang beendet
- Bit 6 = 1 Senden beendet
- Bit 7 = 1 Fehler (allgemein)

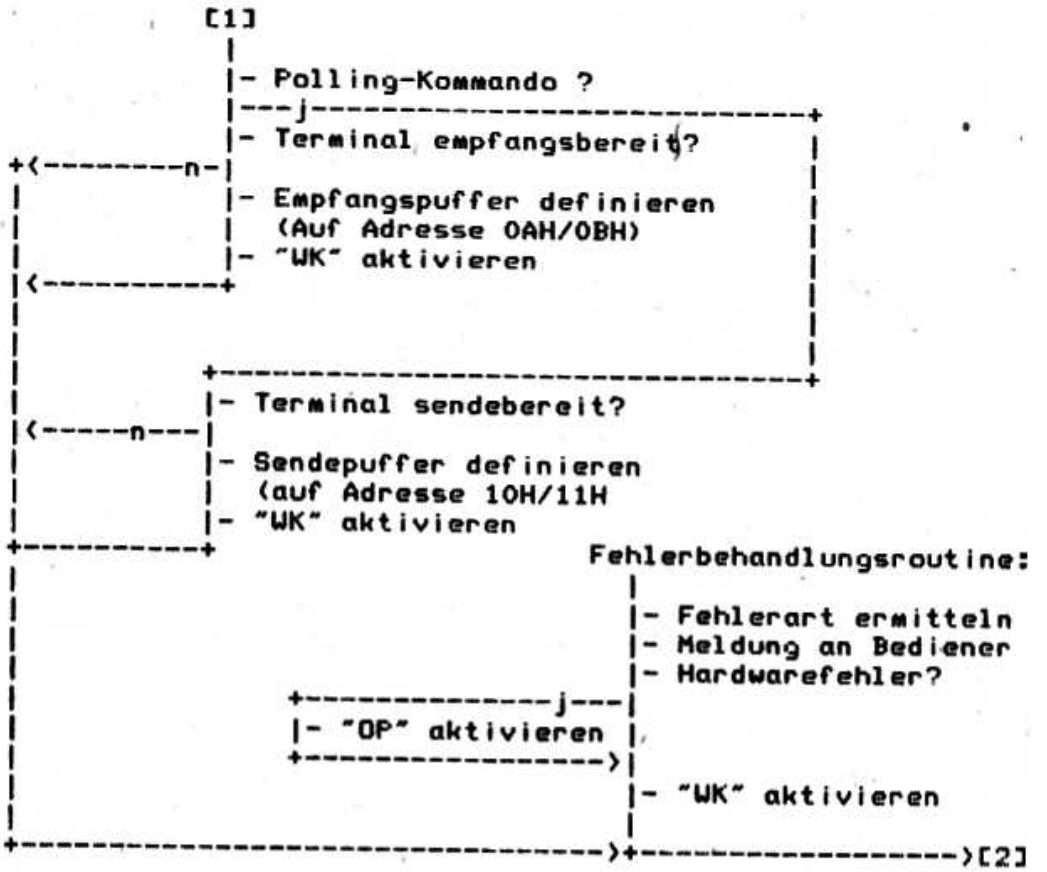
- Statusmeldungen im SEB-Byte des Treibers:

Statuscode	Meldung
H'00'	Text korrekt empfangen
H'20'	Kommandofolge empfangen (nur als Slave)
H'20'	Text fehlerfrei empfangen und anschliessend Textantwort gesendet (nur als Master)
H'40'	Text fehlerfrei gesendet
H'50'	EOT gesendet
H'60'	Text fehlerfrei gesendet, anschliessend fehlerfrei empfangen
H'81'	Ueberlauf Empfangspuffer
H'82'	Abbruch der Uebertragung (EOT im Text empfangen)
H'84'	Abbruch der Uebertragung (16x NAK gesendet)
H'88'	Hardware-Fehler
H'90'	erwarteter Block bzw. Quittung blieb aus
H'C2'	EOT-Quittung empfangen
H'C4'	Abbruch der Uebertragung (Block 16x wiederholt)

Statuscode	Meldung
H'D2'	EOT als Quittung auf eine Kommandofolge empfangen (nur als Master)
H'E2'	Abbruch der Uebertragung, STX-Quittung auf Block empfangen, der nicht der letzte der Nachricht war.

- Beispiel fuer eine Unterbrechungsroutine:





Wird mit gesetztem Bit 2 in CB gearbeitet, ergeben sich Aenderungen:

Wird nicht nach spaetestens 2s bei fehlerfreiem Empfang einer Nachricht das naechste Kommando WK oder RD aktiv, wird die Nachricht positiv quittiert und das Treiberunterprogramm geht wieder in den Steuermodus ueber (Kommandofolge erwarten). Eine danach eintreffende Nachricht wird mit EOT zurueckgewiesen.
Bricht die sendende Station nach Empfang der positiven Quittung auf den letzten Block einer Nachricht die Uebertragung durch Senden von EOT ab, wird der letzte Ruf mit dem Status "Text fehlerfrei empfangen" und BCR = 0 beendet.

Sendet der Treiber Text, wird der entsprechende Ruf nach Empfang der positiven Quittung auf den letzten Block der Nachricht beendet. Mit einem erneuten Ruf kann eine weitere Nachricht gesendet werden. Wird in diesem Ruf als Adresse des Sendepuffers 0H angegeben, wird die Uebertragung durch Senden von EOT abgebrochen und der Ruf mit dem Status "EOT gesendet" beendet. Erfolgt die Aktivierung eines solchen Rufes nicht innerhalb 2s nach Empfang der positiven Quittung, sendet der Treiber automatisch EOT und geht in den Steuermodus.

2. Programmierempfehlungen und Demonstrationsprogramme zur BASIC-Anwendung

2.1. Standardmoduln zur Unterstuetzung der Menuegestaltung

2.1.1. Statisches Menue

Nachfolgende Funktionen sind Grundlage fuer eine einheitliche Programmgestaltung bei der Erzeugung statischer (feststehender) Bildschirmanzeigen:

- Verwenden der Bildschirmsteuerzeichen
- Kursorpositionierung unter Angabe Zeilen-/Spaltenposition
- Festlegen von Druckmasken

Programmierempfehlung:

```
10  REM Bildschirmsteuerzeichen
20  BS$ = CHR$(8)    ' Kursor 1 Zeichen nach links
30  LF$ = CHR$(10)   ' Kursor 1 Zeile nach unten
40  CLR$= CHR$(12)   ' Loeschen Bildschirm und Kursor an Anfang
                       Bildschirm
50  CR$ = CHR$(13)   ' Kursor an Zeilenanfang
60  ES$ = CHR$(20)   ' Loeschen ab Kursorposition bis Ende
                       Bildschirm
70  SK$ = CHR$(21)   ' Kursor 1 Zeichen nach rechts
80  EL$ = CHR$(22)   ' Loeschen ab Kursorposition bis Zeilenende
90  CE$ = CHR$(24)   ' Loeschen Zeile und Kursor an Zeilenanfang
100 LU$ = CHR$(26)   ' Kursor 1 Zeile nach oben
110 SP$ = " "        ' Leerzeichen
120 '
130 '
140 REM Definieren der Funktion fuer die Kursorpositionierung:
        Y = Zeilennummer      1 bis 24
        X = Zeichenposition   1 bis 80
150 DEF FNC$(X,Y) = CHR$(27) + CHR$(X+127) + CHR$(Y+127)
160 '
170 '
180 REM Festlegen von Druckmasken
190 M1$ = "#####": ...
200 '
210 '
220 REM Anwendungsbeispiel
230 REM Loeschen Bildschirm und Positionieren Kursor auf
        Zeile 3, Spalte 5
240 PRINT CLR$ FNC$(3,5)
```

2.1.2. Rollen eines Teiles des Bildschirminhaltes

Es ist ein Teil des Bildschirminhaltes zu rollen. Der verbleibende Rest des Bildschirminhaltes ist feststehend zur Bedienerfuehrung zu nutzen.

- **Assembler-Unterprogramm**

Zur Realisierung der Rollfunktion wird ein in Z80-Assembler-Code geschriebenes Unterprogramm benutzt.
Es werden die Zeilen 2 bis 19 auf die Zeilen 1 bis 18 gerollt.

Assembleruebersetzungsprotokoll:

Assembler-Code	Befehl
21 F850	LD HL, 0F850H ;Adresse 2. Bildschirmzeile
11 F800	LD DE, 0F800H ;Zieladresse Rollen = ;1. Bildschirmzeile
01 05A0	LD BC, 05A0H ;Laenge Rollbereich 18 Zei- ;len * 80 Zeichen
ED B0	LDIR
C9	RET

Die erste Zeile zum Rollen des Bildschirmes wird in HL, die Anfangsadresse des Bildschirmes in DE, die Zeilenanzahl in BC eingetragen. Mit LDIR wird das UP abgearbeitet.

Es ist moeglich, das Unterprogramm getrennt zu erfassen und zu laden. Ebenfalls kann dieses UP in das compilierte BASIC-Programm mit LINK eingebunden werden. Der Unterprogrammcode kann auch in eine DATA-Anweisung eingetragen, mit READ gelesen und mit POKE gezielt in den Speicher abgelegt werden (siehe BASIC-Programm).

Gerufen wird das UP mit den Anweisungen **CALL** oder **USR** (siehe "Anwenderdokumentation BASIC-Interpreter SCPX 1526", Anlage E, Technologie der Unterprogrammarbeit).

- **BASIC-Programm**

Das Assembler-Unterprogramm wird im BASIC-Programm wie folgt aufgerufen (Assembler-Codierungen sind im Beispiel in DATA-Anweisungen abgelegt und sofort verfuegbar):

```
10 REM PROGRAMM ZUM TEILWEISEN ROLLEN DES BILDSCHIRMS
20 '
30 '
40 REM      zu uebergebende Parameter:
50 '      ---------------------------
60 '                keine
70 '
```

```
80 REM       zu uebernehmende Parameter:
90 '        ---------------------------
100 '        UP%   - Startadresse fuer Assembler-UP
110 '
120 REM       interne Parameter:
130 '       --------------------
140 '        H%,H,HI
150 '
160 REM Vorbereitung im Hauptprogramm
161 '
170 '        Festlegen Endadresse fuer BASIC-System
180 CLEAR,&H9000
190 '
191 '
200 REM Laden des Assembler-UP:
    Lesen der Codierungen aus der DATA-Anweisung
    und Ablegen ab Adresse &H9001
210 '
220 H=&H9000
230 RESTORE 250: FOR HI=1 TO 12:
    READ H%: POKE (H+HI),H%: NEXT HI
240 UP%=&H9001                          'Adresse UP
241 '
242 '
245 REM       Codierungen fuer Assembler-UP:
250 DATA 33,80,248,
         17,0,248,
         1,160,5,
         237,176,
         201
260 '
270 REM       Beispiel zur Anwendung
280 '
281 '
282 REM       Bildschirm loeschen
290 PRINT CHR$(12)
291 '
292 '
293 REM       Konstanter Teil fuer Bedienerfuehrung
294 '
295 '
300 I%=19: K%=1: GOSUB 360: PRINT STRING$(79,61)
310 I%=21: K%=1 : GOSUB 360:
    PRINT "FREIER TEIL ZUR BEDIENERFUEHRUNG"
311 '
312 '
313 REM Aufruf UP Rollen Bildschirm (Zeile 1 - 18)
320 CALL UP%
321 '
322 '       Loeschen Zeile 18 auf Leerzeichen
330 I%=18: K%=1: GOSUB 360: PRINT SPACE$(79)
331 '
332 '.  Anzeige "TEXT: " und Aufforderung zur Texteingabe
340 I%=19: K%=1: GOSUB 360: INPUT "TEXT: ",P$
342 '
```

```
344 ' Nach Eingabe Text Verzweigen zum Rollen der zuletzt
      eingegebenen Zeile
350 GOTO 320
352 '
356 REM Funktion zur Kursorpositionierung:
            I%=Zeile,  K%=Spalte
360 PRINT CHR$(27)+CHR$(127+I%)+CHR$(127+K%);
370 RETURN
380 END
```

2.1.3. Maskierte Eingabe

Es wird eine maskierte Eingabe auf dem Bildschirm ueber ein Unterprogramm ermoeglicht, wobei die Maske, die Position der Eingabe und die Anzahl der Punkte festzulegen sind. Der Punkt wird nicht mit eingegeben.

Bevor das UP "Maskierte Eingabe" angesprungen wird, sind Zeile und Spalte fuer die Eingabe, die Eingabemaske sowie die darin maximal enthaltenen Punkte zu definieren.

Im Unterprogramm erfolgen verschiedene Eingabekontrollen:

- Warten bis eine Taste betaetigt wird.
  ```
  1240 H$=INKEY$: IF H$="" THEN GOTO 1240
  ```

- Wird eine Taste mit dem Code <48 oder >57 (Ziffer 0 bis 9) betaetigt, wird die Eingabe ignoriert.
  ```
  1270 IF ASC(H$)<48 OR ASC(H$) >57 THEN GOTO 1230
  ```

- Bei CE-Tastenbetaetigung (Code 24) wird die Eingabe geloescht. Es kann neu eingegeben werden.
  ```
  1250 IF ASC(H$)=24 THEN H3$="": GOTO 1170
  ```

- Bei <ENTER>-Betaetigung (Code 13) wird die Eingabe beendet und das Unterprogramm verlassen.
  ```
  1260 IF ASC(H$)=13 THEN GOTO 1400
  1400 RETURN
  ```

- Die maximale Eingabelaenge entspricht der Laenge der Eingabemaske.

- Quellprogrammliste

```
10 REM BEISPIEL FUER MASKIERTE EINGABE
30 '
40 REM      zu uebergebende Parameter:
50 '     ---------------------------
60 '       SP%      - Positionieren Spalte
70 '       ZE%      - Positionieren Zeile
80 '       MASK$    - Eingabemaske
90 '       HV%      - Max.vorkommende Anzahl der Formatzeichen +1
                      (im Beispiel: Punkte)
100 '
```

```
110 REM     interne Parameter:
120 '     -------------------
130 '     H%,H1%,H2%,H3%,HI%,H%(),H$,H1$,H2$,H3$
140 '
150 '
160 PRINT CHR$(12)
161 '
162 '
163 REM                 Anwendungsbeispiel
164 '
165 REM    1. Eingabe mit Maske
170 HV%=3
180 ZE%=2: SP%=40
190 MASK$="##.##.##"
200 PRINT "Datum: ";
210 GOSUB 1170: PRINT: PRINT
220 '
230 REM    2. Eingabe mit Maske, wobei die Maske veraendert
          wurde.  Die Spaltenposition bleibt erhalten.
240 ZE%=4
250 MASK$="#####"
260 PRINT "Kontonummer: ";
270 GOSUB 1170
280 '
290 REM    3. Eingabe usw.
300 '
310 '
320 '
330 '
340 '
1100 REM      UNTERPROGRAMM FUER MASKIERTE EINGABE
1110 '
1120 '
1130 '      Anzeige Maske auf definierter Zeile / Spalte
1170 H%=0: H3%=SP%: GOSUB 1340: PRINT MASK$
1171 '
1172 '
1173 '      Eingabe in Maske
1174 '
1175 ' Ermittlung der Punkte in der Maske
       (Positionen stehen in Feld H%)
1177 ' H2% = Laenge der Maske mit Formatzeichen
1178 ' H3% = Endeposition Eingabefeld -1
1179 '
1180 H1$="": H2$=".": H1%=1: H2%=LEN(MASK$)
1190 FOR HI%=1 TO HV%
1200 H%(HI%)=INSTR(H1%,MASK$,H2$):
     IF H%(HI%)=0 THEN H1$=H1$+RIGHT$(MASK$,H2%-H%(HI%-1)):
     GOTO 1230
1210 H1%=H%(HI%)+1:
     H1$=H1$+MID$(MASK$,H%(HI%-1)+1,H%(HI%)-H%(HI%-1)-1)
1220 NEXT HI%
1230 H3%=SP%+H2%-1: GOSUB 1340: H$=""
1231 '
1232 '
1233 REM Eingabe der Ziffer mit Eingabekontrollen
```

```
1234 '       In H$ steht eingegebenes Zeichen
1235 '
1240 H$=INKEY$: IF H$="" THEN GOTO 1240
1250 IF ASC(H$)=24 THEN H3$="": GOTO 1170
1260 IF ASC(H$)=13 THEN GOTO 1400
1270 IF ASC(H$)<48 OR ASC(H$)>57 THEN GOTO 1230
1280 IF H%=LEN(H1$) THEN GOTO 1240 ELSE H%=H%+1:
     H1$=H1$+H$: H1$=RIGHT$(H1$,LEN(H1$)-1): H3$=""
1281 '
1282 '
1283 REM Anzeige der eingegebenen Zahl in aufbereiteter Form
         nach jeder Zifferneingabe
1284 '
1285 ' Verschieben Ziffernfolge um 1 Stelle nach links mit
       Beruecksichtigung Formatzeichen und Eintragen letzte
       eingetastete Ziffer an  1. Stelle von rechts
1286 '     H1$ = eingegebene Zahl ohne Formatzeichen
1287 '     H3$ = eingegebene Zahl einschliesslich Formatzeichen
1288 '
1289 '
1290 FOR HI%=1 TO HV%
1300    IF H%(HI%)=0 THEN H3$=H3$+RIGHT$(H1$,H2%-H%(HI%-1)):
        GOTO 1330
1310    H3$=H3$+MID$(H1$,H%(HI%-1)+(2-HI%),
           H%(HI%)-H%(HI%-1)-1)+H2$
1320 NEXT HI%
1330 H3%=SP%: GOSUB 1340: PRINT H3$: GOTO 1230
1331 '
1332 '
1333 REM Funktion zur Kursorpositionierung
1340 PRINT CHR$(27)+CHR$(127+ZE%)+CHR$(127+H3%);: RETURN
1350 '
1390 REM Ruecksprung aus Unterprogramm
1400 RETURN
```

Das Abbrechen des BASIC-Programmes zum teilweisen Rollen des Bildschirmes ist mit ^C moeglich.

2.2. Sortierprogramme

2.2.1. Sortierprogramm 1

- Beschreibung

Der zugrundeliegende Algorithmus sortiert eingegebene Zahlen in absteigender Folge vom Typ REAL (einfache Genauigkeit). Der Programmalgorithmus wird anhand des nachfolgenden einfachen Programms erlaeutert. Durch die Verwendung nur eines Feldes, in dem die Sortierung stattfindet, ist dieses Verfahren sehr speicherplatzsparend. Bei einem verfuegbaren Anwenderspeicher von 27 KByte koennen mit Hilfe des Programmes ca. 6600 Zahlen geordnet werden.

Die Laufzeit fuer 100 zu sortierende Zahlen betraegt fuer ein interpretiertes Programm 75s und ein compiliertes 24s.

Bemerkung:

Soll dieses oder eines der beiden folgenden Programme compiliert werden, ist aufgrund der Sprachunterschiede zwischen Interpreter- und Compilersprache eine Aenderung erforderlich. Fuer die Variable N% ist ein fester Wert vorzugeben. Damit ist jedoch keine freie Auswahl der Anzahl der zu sortierenden Zahlen nach dem Compilieren und Linken moeglich!

- <u>Programmliste</u>

```
10 REM*************************************************************
20 REM***                                                       ***
30 REM***        " S O R T I E R P R O G R A M M  1 "           ***
40 REM***                                                       ***
50 REM*************************************************************
60 PRINT
70 '
80 '
90 REM ANZAHL DER ZU SORT. ZAHLEN UEBER TASTATUR EINGEBEN
100 INPUT "ANZAHL DER ZU SORT. ZAHLEN: ",N%
110 IF N%=0 THEN PRINT "KEINE ZAHLEN ZU ORDNEN!":GOTO 390
120 PRINT
130 DIM A(N%)
140 PRINT
150 PRINT
160 PRINT "ZU SORT. ZAHLEN: "
170 PRINT
180 '
190 '
200 REM ZAHLEN UEBER TASTATUR EINGEBEN
210 FOR I%=1 TO N%
220 INPUT A(I%)
230 NEXT I%
240 '
250 '
260 REM EINGABETEIL BEENDET,UP-AUFRUF
270 '
280 GOSUB 420
290 '
300 REM*************************************************************
310 '
320 REM AUSGABE DER GEORDNETEN ZAHLEN (AUF DRUCKER)
330 LPRINT "ZAHLEN GEORDNET: "
340 LPRINT
350 FOR I%=1 TO N%
360 LPRINT, A(I%)
370 NEXT I%
380 '
390 END
400 '
410 REM*************************************************************
420 '
```

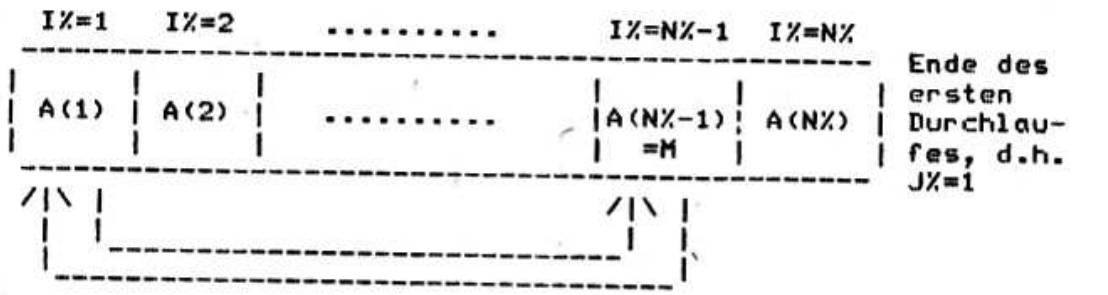
```
430 REM***            UNTERPROGRAMM  S O R T 1
440 REM***            ----------------------
450 'UEBERGABEVARIABLEN:
460 '          N%    :ANZAHL DER ZAHLEN
470 '          A(N%):EINDIM. UNGEORDNETES ZAHLENFELD
480 '          I%    :LAUFVARIABLE FUER INDEX (VON 1...N%)
490 'RUECKGABEVARIABLEN:
500 '          A(N%):GEORDNETES ZAHLENFELD
505 '          I%    :INDEX BZW. POSITION DER JEWEILIGEN ZAHL
510 'INTERNE VARIABLEN:
530 '          T%:INDEX DES MAXIMUMS
540 '          J%:ZAEHLVARIABLE
520 '          M%:MAXIMUM
550 '
560 '                 ----------------------
570 PRINT
580 PRINT "S O R T I E R V O R G A N G   L A E U F T ! "
590 PRINT
600 LET T%=1
610 LET J%=1
620 '
630 '
640 REM ANFANGSWERT FUER MAXIMUM (M) FESTLEGEN
650 LET M=A(J%)
660 '
670 '
680 REM GROESSTE ZAHL ERMITTTELN
690 FOR I%=J% TO N%
700 IF A(I%)>M THEN M=A(I%) :T%=I%
710 NEXT I%
720 '
730 '
740 REM MAXIMUM MIT DER ZAHL A(J%) AUSTAUSCHEN
750 IF M<>A(J%) THEN SWAP A(T%),A(J%)
760 LET J%=J%+1
770 '
780 '
790 REM RUECKSPRUNG, WENN NOCH NICHT ALLE ELEMENTE SORTIERT
800 IF J%<N% THEN  650
810 '
820 '

830 REM ZURUECK ZUM HAUPTPROGRAMM
840 RETURN
850 REM*************************************************************
```

- Erlaeuterungen zum Programmablauf

Alle eingegebenen Zahlen A(1)...A(N%) werden in ein eindimensionales Feld der Groesse N% eingelesen. N% gibt die Anzahl der Feldelemente an. Die Anzahl der zu sortierenden Zahlen N% ist vom Bediener einzugeben. Bei Ueberschreitung der maximal moeglichen Anzahl der zu sortierenden Zahlen erfolgt die Fehlermeldung "out of memory".

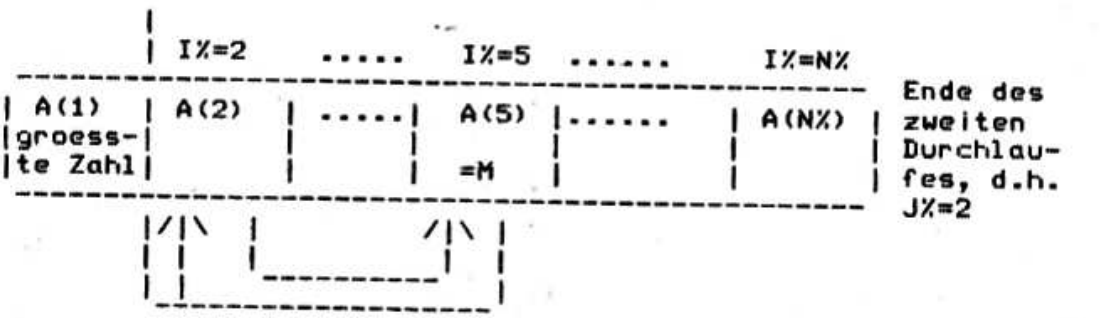
Auf M wird das Maximum ueber die zu durchsuchenden Zahlen abgelegt. Als Anfangswert dafuer wird die erst eingelesene Zahl benutzt. Sie wird mit der nachfolgenden Zahl verglichen. Ist diese groesser, so bekommt das Maximum deren Wert zugeordnet. Die Position des Maximums innerhalb des Feldes wird auf T% vermerkt. Anschliessend erfolgt ein Vergleich mit dem naechsten Feldelement.
Steht auf diese Weise das Maximum aller Zahlen fest und ist es ungleich dem Anfangswert, so erfolgt ein Austausch beider Werte.

Beispiel:



Damit beginnt der zweite Durchlauf, die Variable J% wird um Eins vergroessert. Da die Durchsuchung der Zahlen beim J%-1-ten Element beginnt, wird erst ab der zweiten Zahl mit der Maximumsuche begonnen und das Maximum an die zweite Stelle umgeordnet.

Beispiel:



J% wird wiederum erhoeht, und der naechste Durchlauf beginnt. Wenn J% den Wert von N% erreicht hat, also nur noch eine Zeile zu durchsuchen waere, wird der Algorithmus abgebrochen. Es erfolgt die Ausgabe der geordneten Zahlen als Druckliste.

2.2.2. Sortierprogramm 2

- Beschreibung

Das "Sortierprogramm 2" stellt ein erweitertes Verfahren gegenueber dem vorangegangenen dar. Auch hier koennen etwa 6700 Zahlen sortiert werden. Die Laufzeit ist jedoch wesentlich geringer. Sie betraegt bei 100 Zahlen ca. 35s fuer die Abarbeitung mit dem BASIC-Interpreter und ca. 12s fuer ein compiliertes Programm. Es wird etwa derselbe Speicherplatzbedarf wie beim "Sortierprogramm 1" benoetigt.

- Programmliste

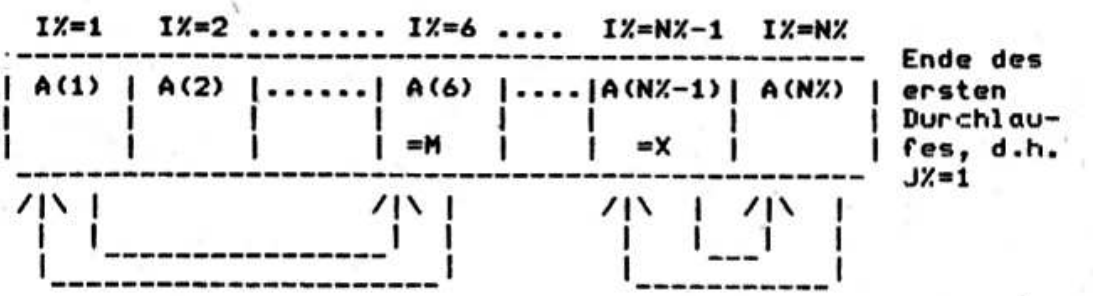
```
10 REM*************************************************************
20 REM***                                                       ***
30 REM***      " S O R T I E R P R O G R A M M 2 "              ***
40 REM***                                                       ***
50 REM*************************************************************
60 '
70 '
80 REM ANZAHL DER ZU SORT. ZAHLEN UEBER TASTATUR EINGEBEN
90 PRINT
100 INPUT"ANZAHL DER ZU SORTIERENDEN ZAHLEN: ",N%
110 IF N%=0  THEN PRINT "KEINE ZAHLEN ZU ORDNEN!":
    GOTO 390
120 DIM A(N%)
130 PRINT
140 PRINT
150 PRINT "ZU SORTIERENDE ZAHLEN: "
160 PRINT
170 '
180 '
190 REM EINGABE DER ZU SORT.ZAHLEN UEBER TASTATUR
200 FOR I%=1 TO N% :INPUT A(I%)
210 NEXT I%
220 PRINT
230 '
240 '
250 PRINT
260 REM EINGABETEIL BEENDET, UP-AUFRUF
270 '
280 GOSUB 400
290 '
300 REM*************************************************************
310 '
320 '
330 REM AUSGABE DER GEORDNETEN ZAHLEN (AUF DRUCKER)
340 PRINT
350 LPRINT "ZAHLEN GEORDNET: "
360 LPRINT
370 FOR I%=1 TO N%:LPRINT USING "####.";I%,:LPRINT A(I%)
380 NEXT I%
390 END
400 REM*************************************************************
```

```
410 '
420 '                  UNTERPROGRAMM S O R T 2
430 '                  -----------------------
440 'UEBERGABEVARIABLEN:
450 '          N%    :ANZAHL DER ZAHLEN
460 '          A(N%):EINDIM. UNGEORDNETES ZAHLENFELD
470 '          I%    :LAUFVARIABLE FUER INDEX
480 'AUSGABEVARIABLEN:
490 '          A(N%):GEORDNETES ZAHLENFELD
500 '          I%    :INDEX BZW. POSITION DER JEWEILIGEN ZAHL
510 'INTERNE VARIABLEN:
520 '          M :MAXIMUM
530 '          T%:INDEX DES MAXIMUMS
540 '          X%:MINIMUM
550 '          S%:INDEX DES MINIMUMS
560 '          J%:ZAEHLVARIABLE
570 '
580 '                  -----------------------
590 S%=1
600 T%=1
610 J%=1
620 PRINT "S O R T I E R V O R G A N G   L A E U F T !"
630 PRINT
640 '
650 '
660 REM ANFANGSWERTE FUER MINIMUM (X) UND MAXIMUM (M) SETZEN
670 M=A(J%)
680 X=A(J%)
690 '
700 '
710 REM ERMITTLUNG VON M UND X
720 FOR I%=J% TO N%-J%+1
730 IF A(I%)>M THEN M=A(I%):T%=I%
740 IF A(I%)<X THEN X=A(I%):S%=I%
750 NEXT I%
760 '
770 '
780 REM GROESSTE ZAHL UND KLEINSTE ZAHL UMORDNEN
790 IF M=A(N%-J%+1)AND X=A(J%) THEN SWAP A(N%-J%+1),A(J%):
     GOTO 830
800 IF M<>A(J%) AND X<>A(J%) THEN SWAP A(T%),A(J%)
810 IF M<>A(J%) AND X=A(J%) THEN SWAP A(J%),A(T%) :
    GOTO 670
820 IF X<>A(N%-J%+1) THEN SWAP A(S%),A(N%-J%+1)
830 J%=J%+1
840 IF J%<=N%/2 THEN GOTO 670
850 PRINT
860 '
870 REM ZURUECK ZUM HAUPTPROGRAMM
880 RETURN
890 REM************************************************************
```

- Erlaeuterungen zum Programm

Im Unterschied zum vorangegangenen Sortierverfahren werden hier in einem Zyklus gleichzeitig die groesste und die kleinste Zahl des eingelesenen Feldes bestimmt. Zum gemeinsamen Anfangswert fuer Minimum X und Maximum M wird die zuerst eingelesene Zahl verwendet. Die Ermittlung von M und X erfolgt analog zum "Sortierprogramm 1". Die Position des gefundenen Minimums im Feld wird in S%, und die Position des Maximums in T% gespeichert. Danach erfolgt die Umordnung des Maximums M an die erste Stelle und die des Minimums X an die letzte Stelle des Feldes.
Diese Umordnung wird durch Tauschen mit den dort angeordneten Feldelementen realisiert, wobei diese im Feld gemaess der vermerkten Position S% bzw. T% transportiert werden.

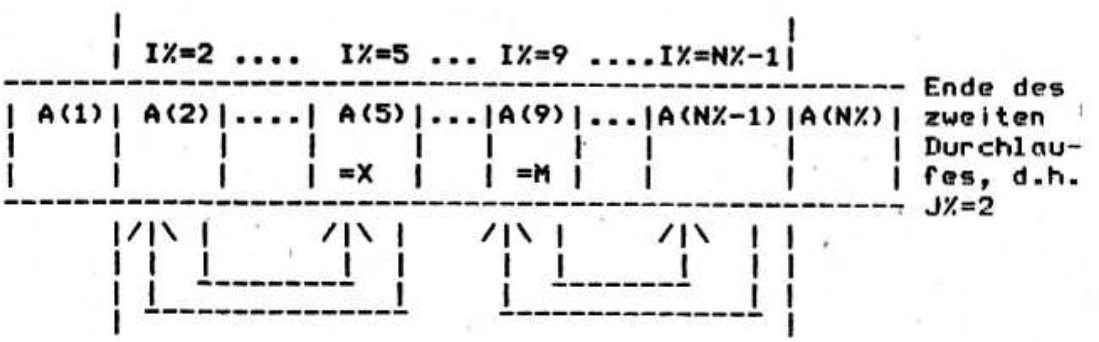
Beispiel:



Bezueglich der Anordnung von Minimum und Maximum koennen einige Faelle auftreten, die gesondert behandelt werden muessen, um keinen fehlerhaften Tausch zu erreichen.
(Beispiel: Minimum an erster Stelle und Maximum an beliebiger Stelle im Feld)
Deshalb sind je nach Anordnung der Zahlen vier bedingte Anweisungen fuer den Tausch vorgesehen.

Nach dem ersten Durchlauf wird J% auf Zwei erhoeht. Das bedeutet hier, dass alle Zahlen von I%=J%=2 bis I%=N%-J%+1 (vorletzte Zahl) durchmustert werden. Der Tausch erfolgt wie schon beschrieben.

Beispiel:



Zum Abbruch dieses Verfahrens kommt es, wenn J%>N%/2 ist, d.h. alle Zahlen liegen geordnet vor. Nachfolgend wird in diesem Programmbeispiel die Druckausgage der geodneten Zahlenfolge realisiert.

2.2.3. Sortierprogramm 3

- Beschreibung

Das dritte Sortierprogrammbeispiel demonstriert eine speicherinterne Sortiermoeglichkeit fuer Zeichenketten in aufsteigender Folge. Hierfuer wurde das eben beschriebene Programm modifiziert. Dabei wird von dem in BASIC moeglichen Zeichenkettenvergleich Gebrauch gemacht.
Die Zeichen einer Zeichenkette werden vom Interpreter von links nach rechts verglichen. Die Codierung des ersten nicht uebereinstimmenden Zeichens bestimmt, welcher Kettenwert groesser ist (laut Codetabellenwert). Ist z.B. eine Zeichenkette die Teilkette einer zweiten, so wird die Teilkette als kleinerer Wert festgestellt.

- Programmliste

```
10 REM*************************************************************
20 REM***                                                       ***
30 REM***       " S O R T I E R P R O G R A M M 3 "             ***
40 REM***       ( ZUM SORTIEREN VON ZEICHENKETTEN )             ***
50 REM***                                                       ***
60 REM*************************************************************
70 '
80 '
90 REM ANZAHL DER ZU SORT. ZEICHENKETTEN UEBER TASTATUR
   EINGEBEN
100 PRINT
110 INPUT "ANZAHL DER ZU SORTIERENDEN ZEICHENKETTEN : ",N%
120 IF N%=0  THEN PRINT "KEINE ZEICHENKETTEN ZU ORDNEN!":
    GOTO 420
130 DIM A$(N%)
140 PRINT
150 PRINT
160 PRINT "ZU SORTIERENDE ZEICHENKETTEN : "
170 PRINT
180 '
190 '
200 REM EINGABE DER ZU SORT.ZEICHENKETTEN UEBER TASTATUR
210 FOR I%=1 TO N% :LINE INPUT A$(I%)
220 NEXT I%
230 PRINT
240 '
250 '
260 PRINT
270 REM EINGABETEIL BEENDET, UP-AUFRUF
280 '
290 GOSUB 440
```

```
300 '
310 REM*************************************************************
320 '
330 '
340 REM AUSGABE DER GEORDNETEN ZEICHENKETTEN (AUF DRUCKER)
350 PRINT
360 LPRINT "ZEICHENKETTEN  GEORDNET : "
370 LPRINT
380 FOR I%=1 TO N%:LPRINT USING "####.";I%,:LPRINT A$(I%)
390 NEXT I%
400 '
410 '
420 END
430 '
440 REM*************************************************************
450 '
460 '                  UNTERPROGRAMM S O R T 3
470 '                  ----------------------
480 'UEBERGABEVARIABLEN:
490 '          N%     :ANZAHL DER ZEICHENKETTEN
500 '          A$(N%):EINDIM. UNGEORDNETES ZEICHENKETTENFELD
510 '          I%     :LAUFVARIABLE FUER INDEX (VON 1...N%)
520 'RUECKGABEVARIABLEN:
530 '          A$(N%):GEORDNETES ZEICHENKETTENFELD
540 '          I%     :INDEX BZW.POSITION DER JEWEILIGEN ZK
550 'INTERNE VARIABLEN:
560 '          M :MAXIMUM
570 '          T%:INDEX DES MAXIMUMS
580 '          X :MINIMUM
590 '          S%:INDEX DES MINIMUMS
600 '          J%:ZAEHLVARIABLE
610 '
620 '                  ----------------------
630 REM ES BEGINNT SORTIERVORGANG
640 T%=1
650 J%=1
660 S%=1
670 PRINT "S O R T I E R V O R G A N G   L A E U F T !"
680 PRINT
690 '
700 '
710 REM ANFANGSWERTE FUER MINIMUM (X$) UND MAXIMUM (M$) SETZEN
720 M$=A$(J%)
730 X$=A$(J%)
740 '
750 '
760 REM ERMITTLUNG VON M$ UND X$
770 FOR I%=J% TO N%-J%+1
780 IF A$(I%)>M$ THEN M$=A$(I%):T%=I%
790 IF A$(I%)<X$ THEN X$=A$(I%):S%=I%
800 NEXT I%
810 '
820 '
830 REM GROESSTE UND KLEINSTE ZEICHENKETTE UMORDNEN
840 IF X$=A$(N%-J%+1) AND M$=A$(J%) THEN
    SWAP A$(N%-J%+1),A$(J%): GOTO 880
```

```
850 IF M$<>A$(J%) AND X$<>A$(J%)THEN SWAP A$(S%),A$(J%)
860 IF X$<>A$(J%) AND M$=A$(J%) THEN
    SWAP A$(J%),A$(S%):GOTO 720
870 IF M$<>A$(N%-J%+1) THEN SWAP A$(T%),A$(N%-J%+1)
880 J%=J%+1
890 IF J%<=N%/2 THEN GOTO 720
900 PRINT
910 '
920 '
930 REM ZURUECK ZUM HAUPTPROGRAMM
940 RETURN
950 REM************************************************************
```

- **Beispiel zum Zeichenketten-Sortierprogramm**

- Vornamenverzeichnis:

```
run

ANZAHL DER ZU SORTIERENDEN ZEICHENKETTEN : 7
ZU SORTIERENDE ZEICHENKETTEN :

KATRIN
karin
anne
annekatrin
karina
anne-katrin
katrin

S O R T I E R V O R G A N G  L A E U F T  !

ZEICHENKETTEN GEORDNET :

1. KATRIN
2. anne
3. anne-katrin
4. annekatrin
5. karin
6. karina
7. katrin
OK
```

2.3. Verarbeitung grosser Diskettendatenmengen

Anhand des nachfolgenden Programmbeispieles soll die Zugriffsmoeglichkeit von BASIC-Programmen auf grosse Diskettendatenmengen erlaeutert werden.
Die Arbeit mit grossen Dateien im sequentiellen Zugriff ist zeitaufwendig und uneffektiv. Folglich wird die Erlaeuterung auf Direktzugriffsdateien beschraenkt.
Es wird davon ausgegangen, dass sowohl die Verarbeitung der Datei auf einem, als auch nacheinander auf mehreren Laufwerken moeglich ist sein muss.

- Programmalgorithmus

1. Das Aufzeichnen auf eine Diskette ist nach Diskettenwechsel ohne Fehlermeldung

   Bdos Err On X: R/O

   moeglich mit der BASIC-Anweisung RESET:

```
  70 RESET
1510 RESET
```

2. Es sind die zur Eroeffnung der Datei notwendigen Variablen festzulegen:

```
90 L$= "B:"            Datei wird im LW B eroeffnet.
   D$= "MULT.ASC"      Dateiname
   Z=  16              Satzlaenge
```

   Diese Variablen werden wiederholt bei OPEN verwendet.

```
150  OPEN "R", #1, L$+D$,Z
1390 OPEN "R", #1, L$+D$,Z
1520 OPEN "R", #1, L$+D$,Z
```

3. Das Eroeffnen der Direktzugriffsdatei erfolgt mittels Anweisungszeile 150. Nachfolgend wird der erste Satz der Datei eingelesen.

```
150 OPEN "R",#1, L$+D$,Z
160 SATZNR=1
170 FIELD #1, 2 AS SNR$, 2 AS SA$, 2 AS LW$,
    8 AS DT$, 2 AS VW$
180 GET #1, SATZNR
190 SNR%=CVI(SNR$): VW%=CVI(VW$): SA%=CVI(SA$)
```

   Der erste Satz der Datei enthaelt die Angaben:

   - aktuelle Satznummer der Diskette SNR%
   - aktuelle Satznummer der Gesamtdatei SA%
   - Dateiname
   - ein Dateiverweis VW% (VW% = 1 Teildatei, VW% = 2 Dateiende).

Diese Informationen werden angezeigt mit

```
270 IF VW% =0 THEN GOTO 330
280 PRINT "DATEINAME: ";LW$.DT$: PRINT
290 PRINT "AKTUELLE"  TAB(30) "AKTUELLE "
300 PRINT
    "SATZNR. DISKETTE :  ";SNR% -1
    TAB(30) "SATZNR. GESAMTDATEI:   ";SA%
    TAB(63) "VERWEIS: ";VW%
310 PRINT


320 IF VW%= 1 THEN PRINT
    "1 = TEILDATEI ---> NEUE DISKETTE EINLEGEN!":
    GOTO 340


330 IF VW%= 2 THEN PRINT
    "2 = DATEIENDE ERREICHT  -
    DATEI KANN FORTGESCHRIEBEN WERDEN!"
    ELSE PRINT "NEUE DATEI -  LESEN NICHT MOEGLICH!"
```

4. Handelt es sich um eine 'leere' Datei, d.h., es sind noch keine Saetze gespeichert, wird als Satznummer = 2 eingestellt. Das Schreiben von Saetzen kann beginnen.
Sonst wird mit der aktuellen Satznummer (SNR%) fortgesetzt.

```
390 IF EOF(1) THEN GOTO 400 ELSE GOTO 410
400 SNR%=2: SA%=1
410 FIELD #1, 2 AS PSA$, 12 AS PMATNR$
730 SA%=SA%+1
740 LSET PSA$=MKI$(SA%): LSET PMATNR$=MATNR$
750 PUT #1, SNR%
790 SNR%=SNR%+1
```

5. Ist die Zahl der aufzuzeichnenden Saetze erreicht, werden in den ersten Satz der Datei die aktuellen Dateiinformationen

   - aktuelle Satznummer Diskette (SNR%)
   - aktuelle Satznummer Gesamtdatei (SA%)
   - Dateiname
   - Dateiverweis = 2

   geschrieben und die Datei geschlossen.

```
870 FIELD #1, 2 AS SNR$,  2 AS SA$,
    2 AS LW$, 8 AS DT$, 2 AS VW$
880 SATZNR=1: V%=2
890 LSET SNR$=MKI$(SNR%): LSET SA$=MKI$(SA%):
    LSET LW$=L$: LSET DT$=D$: LSET VW$=MKI$(V%)
900 PUT #1, SATZNR
910 CLOSE #1
```

6. Eine Teildatei wird gelesen mittels

```
1070 FOR I%=2 TO SNR%-1
1090 GET #1, I%
1100 PRINT
     "SATZNUMMER GESAMTDATEI: " CVI(PSA$)
     TAB(40) "MATERIALNUMMER = "; PMATNR$
1110 NEXT
```

7. Fehlerbehandlung

Die Fehlerbehandlungsroutine wird bei Auftreten eines Fehlers angesprungen durch

80 ON ERROR GOTO 1200.

In dieser Fehlerbehandlungsroutine wird der

Fehlercode 61 = Diskette voll

getrennt behandelt, da hier die Aufzeichnung auf einer weiteren Diskette fortgesetzt werden soll.

. Wird der Fehlercode 61 erkannt, wird die aktuelle Datei automatisch geschlossen. Es ist ein erneutes OPEN auf die aktuelle Datei erforderlich. Anschliessend wird der erste Satz der Datei aktualisiert, d.h.,

- Satzzaehler
- Dateiverweis = 1 --> Teildatei

werden eingetragen. Die Datei wird geschlossen.

```
1390 OPEN "R", #1, L%+D%,Z
1400 FIELD #1, 2 AS SNR$,  2 AS SA$, 2 AS LW$,
     8 AS DT$, 2 AS VW$
1410 SATZNR=1: V%=1
1420 SA%=SA%-1
1430 LSET SNR$=MKI$(SNR%): LSET SA$=MKI$(SA%):
     LSET LW$=L$: LSET DT$=D$:LSET VW$=MKI$(V%)
1440 PUT #1, SATZNR
1450 CLOSE #1
```

. Nach Diskettenwechsel und Bedienen von <ENTER> wird die aktuelle Datei fuer die gewechselte Diskette erneut eroeffnet.
Die Datei wird fortgeschrieben, wobei mittels Anweisungszeile 1570 der Satz geschrieben wird, welcher die Meldung "Diskette voll" provoziert hatte.

```
1500 INPUT "",I
1510 RESET
1520 OPEN "R", #1, L%+D%,Z
1530 SA%=SA%+1
1540 FIELD #1, 2 AS PSA$, 12 AS PMATNR$
1550 LSET PSA$=MKI$(SA%): LSET PMATNR$= MATNR$
```

```
1560 SNR%=2
1570 PUT #1, SNR%
1580 RESUME
```

Soll im Multivolumebetrieb mit mehreren Laufwerken gearbeitet werden, muss in Zeile 1520 ein OPEN auf das entsprechende Laufwerk erfolgen.

Beispiel:

```
1520 OPEN "R", #1, "C:"+D$,Z
```

- **Quellprogrammliste**

```
10 REM PROGRAMMIERBEISPIEL ZUR VERARBEITUNG GROSSER
       DISKETTENDATENMENGEN
20 '
30 '
40 REM     PROGRAMMVORBEREITUNG
50 '
60 PRINT CHR$(12)                          ' Bildschirm loeschen
70 RESET                                   ' Laufwerk zur Auf-
                                             zeichnung bereit
80 ON ERROR GOTO 1200                      ' Sprung an
                                             Fehlerbehandlung
90 L$="B:": D$="MULT.ASC": Z=16            ' Variable fuer
                                             Dateieroeffnung
100 '
110 '
120 '
130 REM     EROEFFNEN DATEI UND EINLESEN 1. SATZ
140 '
150 OPEN "R",#1, L$+D$,Z
160 SATZNR=1
170 FIELD #1, 2 AS SNR$, 2 AS SA$, 2 AS LW$,
              8 AS DT$, 2 AS VW$
180 GET #1, SATZNR
190 SNR%=CVI(SNR$): VW%=CVI(VW$): SA%=CVI(SA$)
200 '
210 '
220 PRINT TAB(20) "TESTPROGRAMM  MULTIVOLUMEBETRIEB"
230 PRINT TAB(20) STRING$(32,"_")
240 PRINT: PRINT
250 '
260 '
270 IF VW% =0 THEN GOTO 330
280 PRINT "DATEINAME: ";LW$ DT$: PRINT
290 PRINT "AKTUELLE"  TAB(30) "AKTUELLE "

300 PRINT    "SATZNR. DISKETTE :  ";SNR% -1
             TAB(30) "SATZNR. GESAMTDATEI:    ";SA%
             TAB(63) "VERWEIS: ";VW%
310 PRINT
320 IF VW%= 1 THEN PRINT
```

```
    "1 = TEILDATEI ---> NEUE DISKETTE EINLEGEN!":
    GOTO 340
330 IF VW%= 2 THEN PRINT
    "2 = DATEIENDE ERREICHT  -
    DATEI KANN FORTGESCHRIEBEN WERDEN!"
    ELSE PRINT "NEUE DATEI -  LESEN NICHT MOEGLICH!"
340 PRINT
350 '
360 '
370 '   ERMITTELN, OB ES SICH UM NEUE DATEI HANDELT,
        SOWIE VORBEREITUNG FUER AUFZEICHNEN DER DATEI
380 '
390 IF EOF(1) THEN GOTO 400 ELSE GOTO 410
400 SNR%=2: SA%=1
410 FIELD #1, 2 AS PSA$, 12 AS PMATNR$
420 '
430 '
440 '
450 '
460 REM   PROGRAMMAUSWAHL
470 '
480 PRINT "PROGRAMMAUSWAHL:"
          TAB(30) "SCHREIBEN AUF DISKETTE = S"
490 PRINT TAB(30) "DISKETTE LESEN         = L"
500 PRINT TAB(30) "PROGRAMMENDE           = E"
510 J$=INPUT$(1)
520 IF J$="E" OR J$="e" THEN END
530 IF J$="L" OR J$="l" THEN GOTO 1020
540 IF VW%=1  THEN PRINT:
    PRINT "TEILDATEI -  SCHREIBEN NICHT MOEGLICH!":
    PRINT "DISKETTE WECHSELN  --> PROGRAMM NEU STARTEN!":
    END
550 IF J$="S" OR J$="s" THEN GOTO 560 ELSE GOTO 60
560 PRINT
570 '
580 '
590 '
600 '
610 '
620 REM   DATEI AUFZEICHNEN
630 '
640 INPUT "ANZAHL DER AUFZUZEICHNENDEN SAETZE: ",A
650 FOR S%= 1 TO A
660 NR#=INT(999999999999#*RND+1)
670 NR$=STR$(NR#)
680 L=LEN(NR$)-1
690 ZL=12-(LEN(NR$)-1)
700 N1$=STRING$(ZL,48)
710 N2$=RIGHT$(NR$,L)
720 MATNR$=N1$+N2$
730 SA%=SA%+1
740 LSET PSA$=MKI$(SA%): LSET PMATNR$=MATNR$
750 PUT #1, SNR%
760 PRINT "LFD. NUMMER =", S%
770 PRINT "SATZNR. GESAMTDATEI:" SA%
          TAB(40) " MATERIALNUMMER = " MATNR$
```

```
780 PRINT
790 SNR%=SNR%+1
800 NEXT
810 '
820 '
830 '
840 REM   ORDNUNGSGEMAESSER ABSCHLUSS DER DATEI
850 '
860 '
870 FIELD #1, 2 AS SNR$,  2 AS SA$,
              2 AS LW$, 8 AS DT$, 2 AS VW$
880 SATZNR=1: V%=2
890 LSET SNR$=MKI$(SNR%): LSET SA$=MKI$(SA%):
    LSET LW$=L$: LSET DT$=D$: LSET VW$=MKI$(V%)
900 PUT #1, SATZNR
910 CLOSE #1
920 STOP
930 GOTO 60
940 '
950 '
960 '
970 '
980 '
990 REM   LESEN DER DATEI
1000 '
1010 '
1020 PRINT CHR$(12)
1030 GET #1,1
1040 IF EOF(1) THEN  PRINT "NEUE DATEI -
                     LESEN NICHT MOEGLICH! ":
     FOR I=1 TO 2000: NEXT: GOTO 60
1050 '
1060 '
1070 FOR I%=2 TO SNR%-1
1080 GET #1, I%
1090 PRINT  "SATZNUMMER GESAMTDATEI: " CVI(PSA$)
            TAB(40) "MATERIALNUMMER = "; PMATNR$
1100 NEXT:STOP
1110 '
1120 '
1130 IF VW%=1 THEN PRINT
     "NEUE DISKETTE EINLEGEN  -  PROGRAMM NEU STARTEN!":
     END
1140 GOTO 60
1150 '
1160 '
1170 '
1180 '
1190 '
1200 REM FEHLERBEHANDLUNG
1210 '
1220 IF ERR=61 THEN GOTO 1330
1230 '
1240 '   SONSTIGE FEHLER WIE Z.B. ERROR 57: DISK I/O ERROR
1250 '
```

```
1260 PRINT
     "KEIN AUFZEICHNEN MOEGLICH. DISKETTE WECHSELN. -->
      PROGRAMM NEU STARTEN!": END
1270 '
1280 '
1290 '
1300 '
1310 '    FEHLER 61 = DISKETTE GEFUELLT
1320 '
1330 PRINT "DISKETTE VOLL. NEUE DISKETTE EINLEGEN.
           < ENTER > BEDIENEN! "
1340 IF SNR%=2 THEN GOTO 1580
1350 '
1360 '
1370 '    AKTUALISIEREN DES 1. SATZES DER DATEI
     UND DATEIABSCHLUSS
1380 '
1390 OPEN "R", #1, L%+D%,Z
1400 FIELD #1, 2 AS SNR%,  2 AS SA%,
               2 AS LW%, 8 AS DT%, 2 AS VW%
1410 SATZNR=1: V%=1
1420 SA%=SA%-1
1430 LSET SNR%=MKI%(SNR%): LSET SA%=MKI%(SA%):
     LSET LW%=L%: LSET DT%=D%:LSET VW%=MKI%(V%)
1440 PUT #1, SATZNR
1450 CLOSE #1
1460 '
1470 '
1480 '    DISKETTENWECHSEL  -- AUFZEICHNEN FORTSETZEN
1490 '
1500 INPUT "",I
1510 RESET
1520 OPEN "R", #1, L%+D%,Z
1530 SA%=SA%+1
1540 FIELD #1, 2 AS PSA%, 12 AS PMATNR%
1550 LSET PSA%=MKI%(SA%): LSET PMATNR%=MATNR%
1560 SNR%=2
1570 PUT #1, SNR%
1580 RESUME
```

2.4. Demonstrationsprogramme

Es werden BASI-Spielprogramme vorgestellt, wobei nicht das Spiel an sich im Vordergrund steht, sondern die Verwendung von BASIC-Ausdrucksmitteln anhand dieser Programme demonstriert werden soll.

2.4.1. TREFFER

- Kurzcharakteristik des Programms

Einige Bemerkungen zu den Spielregeln des TREFFER-Spiels sind zum Verstaendnis notwendig:

Das TREFFER-Spiel beginnt mit der Darstellung eines Zeilen-Spalten-Rasters (Zeilen 0 bis 9 und Spalten A bis S) auf dem Bildschirm. Dies ist die Spielmatrix.
Vor Ausgabe der Matrix wird der Bediener zur Eingabe einer Zahl im Bereich von 1 bis 5000 aufgefordert. Ausgehend von dieser Zahl wird eine zufaellige Belegung von 45 Feldern der insgesamt 190 Felder mit Zahlenwerten vorgenommen. Diese Belegung ist aber fuer den Spieler nicht sichtbar.
Der Spieler muss nun ueber Eingabe von Zeilen- und Spaltenposition das Feld auswaehlen, auf welchem er einen Zahlenwert vermutet. Wurde ein Zahlenwert getroffen, erfolgt seine Anzeige. War kein Zahlenwert enthalten, wird ein *-Zeichen vermerkt.
Zum Abschluss des Spiels wird die Summe der Treffer ausgegeben.

Normalerweise waere dies ein reines Zufallsspiel, bei dem der Anwender nicht vorher wissen kann, wo welche Werte stehen und ob ueberhaupt ein Wert im Feld vorhanden ist.
Um dieses Spiel attraktiver zu gestalten und dem Spieler gewisse Moeglichkeiten zu geben belegte Felder mit einer gewissen Wahrscheinlichkeit voraussagen zu koennen, sind folgende Werteketten (zusammenhaengende Felder mit jeweils gleicher Wertigkeit) implementiert:

2	Ketten	mit	4	Elementen	mit Wert je 3 /Feld
1	Kette	mit	4	"	" " 2 /Feld
3	Ketten	mit	3	"	" " 4 /Feld
4	Ketten	mit	2	"	" " 5 /Feld
5	Ketten	mit	2	"	" " 6 /Feld
6	Ketten	mit	1	"	" " 7 /Feld

Diese Ketten koennen in folgender Form vorkommen:

1. Waagerechte Kette
2. Senkrechte Kette
3. Diagonale nach rechts
4. Diagonale nach links

- BASIC-Ausdrucksmittel, demonstriert am vorliegenden Programm

Nun zu den verwendeten Ausdrucksmitteln in Verbindung mit der Loesung spezieller Aufgaben bei der Abarbeitung des hier vorliegenden Demonstrationsprogrammes:

- Es koennen feststehende Menues /Ausgaben ueber Bildschirm realisiert werden. Dazu wird eine Funktion zur Positionierung des Kursors gebraucht, die der Anwender folgendermassen definiert:

```
1030 DEF FNC$(Y,X) = CHR$(27) + CHR$(Y+127) + CHR$(X+127)
```

- Anzeige der Spielmatrix
  Es wird mit Steuerzeichenausgabe auf Bildschirm gearbeitet, z.B. PRINT CHR$(12) loescht Bildschirm. Analog sind alle anderen Bildschirmsteuerzeichen ausgebbar.
  Mittels Laufanweisungen wird fuer die Ausgabe der Spielfeldmatrix gesorgt (Ausgabe der senkrechten und waagerechten Striche einschliesslich Beschriftung der Zeilen mit Nummern von 0 bis 9 und der Spalten mit den Buchstaben von A bis S.

```
1220 PRINT CHR$(12) CHR$(13) " ";
1230 FOR C%=65 TO 83:PRINT "   " CHR$(C%);:NEXT:PRINT
1240 FOR L%=0 TO 9
1250   PRINT"  |";:FOR C%=1 TO 19:PRINT"---|";:NEXT:PRINT
1260   PRINT USING "# |";L%;:
       FOR C%=1 TO 19:PRINT"   |";:NEXT:PRINT
1270 NEXT:PRINT"  |";:FOR C%=1 TO 19:PRINT"---|";:NEXT
```

- Sollen bestimmte Anweisungsfolgen wiederholt abgearbeitet werden, so bedient man sich Unterprogrammen. Hier ist das Belegen der Matrix mit Werteketten dieser wiederholt ablaufende Prozess. Dabei sind Uebergabewerte (Parameter fuer den Prozess) variabel.
  Der Aufruf eines UP erfolgt ueber GOSUB. Die Werte fuer das UP werden ueber Variable bereitgestellt.

```
1370 K=2:N=1:S=4:GOSUB 1920
1380 K=3:N=2:S=4:GOSUB 1920
1390 K=4:N=3:S=3:GOSUB 1920
1400 K=5:N=4:S=2:GOSUB 1920
1410 K=6:N=5:S=2:GOSUB 1920
1420 K=7:N=6:S=1:GOSUB 1920
```

Das eigentliche UP kann jede beliebige Anweisungsfolge sein, die aber mit RETURN als logisches Ende abgeschlossen sein muss.

```
1920 '
1930 '  Belegung der Spielfeld-Matrix mit Werten
1940 '
```

```
1950 '   ------------------------------------
1960 '   | Matrix           | Matrixelement mit  |
1970 '   |eingestellt       | 1 belegt, wo Spie- |
1980 '   |                  | ler  Wert vermutet |
1990 '   |                  |                    |
2000 '   |                  |                    |
2010 '   ------------------------------------
2020 '
2030 '
2040 U%=U%+N*S:
2050 FOR C%=1 TO N
2052 '
2053 '                Auswahl Startelement (zufallsbedingt)
2054 '
2060   X=INT(RND(1)*19):Y=INT(RND(1)*10):
       IF G(X,Y)<>0 THEN 2060 ELSE G(X,Y)=K
2070   IF S=1  THEN 2420
2071 '
2072 '              AUSWAHL KETTENEINTRAGUNGSRICHTUNG
2073 '                      (zufallsbedingt)
2074 '
2080   W%=0:I%=INT(RND(1)*5+1):
       ON I% GOTO 2100,2120,2180,2240,2300,2360
2090 '
2100   G(X,Y)=0:GOTO 2060
2110 '
2111 '                           Waagerechte Kette
2112 '
2120   IF X+S-1>18 THEN 2080
2130   FOR Z%=1 TO S-1:IF G(X+Z%,Y)=0 THEN W%=W%+1
2140   NEXT
2150   IF W%<>S-1 THEN 2080
2160   FOR Z%=1 TO S-1:G(X+Z%,Y)=K:NEXT:GOTO 2420
2170 '
2171 '                           Senkrechte Kette
2172 '
2180   IF Y+S-1>9 THEN 2080
2190   FOR Z%=1 TO S-1:IF G(X,Y+Z%)=0 THEN W%=W%+1
2200   NEXT
2210   IF W%<>S-1 THEN 2080
2220   FOR Z%=1 TO S-1:G(X,Y+Z%)=K:NEXT:GOTO 2420
2230 '
2231 '                        Diagonale nach rechts unten
2232 '
2240   IF X+S-1>18 OR Y+S-1>9 THEN 2080
2250   FOR Z%=1 TO S-1:IF G(X+Z%,Y+Z%)=0 THEN W%=W%+1
2260   NEXT
2270   IF W%<>S-1 THEN 2080
2280   FOR Z%=1 TO S-1:G(X+Z%,Y+Z%)=K:NEXT:GOTO 2420
2289 '
2290 '                        Diagonale nach rechts oben
2291 '
2300   IF X+S-1>18 OR Y-S-1<0 THEN 2080
2310   FOR Z%=1 TO S-1:IF G(X+Z%,Y-Z%)=0 THEN W%=W%+1
2320   NEXT
2330   IF W%<>S-1 THEN 2080
```

```
2340   FOR Z%=1 TO S-1:G(X+Z%,Y-Z%)=K:NEXT:GOTO 2420
2350 '
2351 '                          Diagonale nach links unten
2352 '
2360   IF X-S-1<0 OR Y+S-1>9 THEN 2080
2370   FOR Z%=1 TO S-1:IF G(X-Z%,Y+Z%)=0 THEN W%=W%+1
2380   NEXT
2390   IF W%<>S-1 THEN 2080
2400   FOR Z%=1 TO S-1:G(X-Z%,Y+Z%)=K:NEXT:GOTO 2420
2410 '
2420 NEXT
2430 RETURN
```

- Eingabeaufforderungen erfolgen stets in Verbindung mit der Anzeige von Bedienertext. Der Ort der Eingabe ist ueber Zeilen-/Spaltenangabe in Verbindung mit Aufruf der Kursorsteuerfunktion an jeder beliebigen Stelle des Bildschirms waehlbar:

```
1550 PRINT FNC$(24,10) "Eingabe Horizontal (A-S)  : ";
```

Die Zahl der einzugebenden Zeichen bei Tastatureingabe kann fest vorgegeben werden. Eine Ueberpruefung der eingegebenen Daten kann mit dem Ziel angeschlossen werden, dass nur zulaessige Zeichen akzeptiert werden, unzulaessige keine Fehlermeldung bewirken, sondern zur erneuten Eingabe auffordern.

```
1560 H$=INPUT$(1):IF H$<"A" OR H$>"S" THEN 1560
```

- Werden Verzoegerungen bei der Abarbeitung des Dialoges gewuenscht, laesst sich dies mittels Laufanweisungen realisieren:

```
1780 FOR C%=1 TO 3000:NEXT
```

- Zufallszahlen aus fest vorgegebenen Wertebereichen werden mittels der Standardfunktionen INT und RND bereitgestellt:

```
2060 X=INT(RND(1)*19):Y=INT(RND(1)*10):
     IF G(X,Y)<>0 THEN 2060 ELSE G(X,Y)=K
```

- Eine Auswahl von Anweisungsfolgen, die in Abhaengigkeit von einer Bedingung getroffen wird, laesst sich mittels der ON ...GOTO-Anweisung realisieren. In diesem Beispiel wird damit die Ketteneintragungsrichtung ausgewaehlt:

```
2080  W%=0:I%=INT(RND(1)*5+1):
      ON I% GOTO 2100,2120,2180,2240,2300,2360
```

- Die Arbeit mit Matrizen bzw. allgemein mit Feldern erfolgt ueber indizierte Variable. Ueber diese Variable wird der Zugriff auf die Feldelemente vorgenommen:

```
Zuweisung eines Wertes auf ein Feldelement/Abfrage des
Wertes eines Feldelementes
2180 IF Y+S-1)9 THEN 2080
2190 FOR Z%=1 TO S-1:IF G(X,Y+Z%)=0 THEN W%=W%+1
2200 NEXT
2210 IF W%()S-1 THEN 2080
2220 FOR Z%=1 TO S-1:G(X,Y+Z%)=K:NEXT:GOTO 2420
```

- **Bedienung**

- Der Bediener wird zur Eingabe einer beliebigen Zahl im Bereich von 1 bis 5000 mittels Ausschrift

```
Eingabe Zahl im Bereich von 1 ... 5000:
```

aufgefordert (Zahl bewirkt zufaellige Belegung der Spielmatrix mit Werten.)

- Die Anzahl der Spielrunden ist einzugeben:

```
Max. Rundenzahl:
```

- Die Spielmatrix wird angezeigt, und pro Spielrunde kann ein Feld ausgewaehlt werden. Dazu wird man ueber die Ausschrift

```
Eingabe Horizontal (A-S):    Vertikal (0-9):
```

aufgefordert.

Nach dieser Eingabe startet das Spiel die Suche in der Matrix. Dies wird sichtbar gemacht ueber die Ausschriften

```
Start    und    ** Suche ** .
```

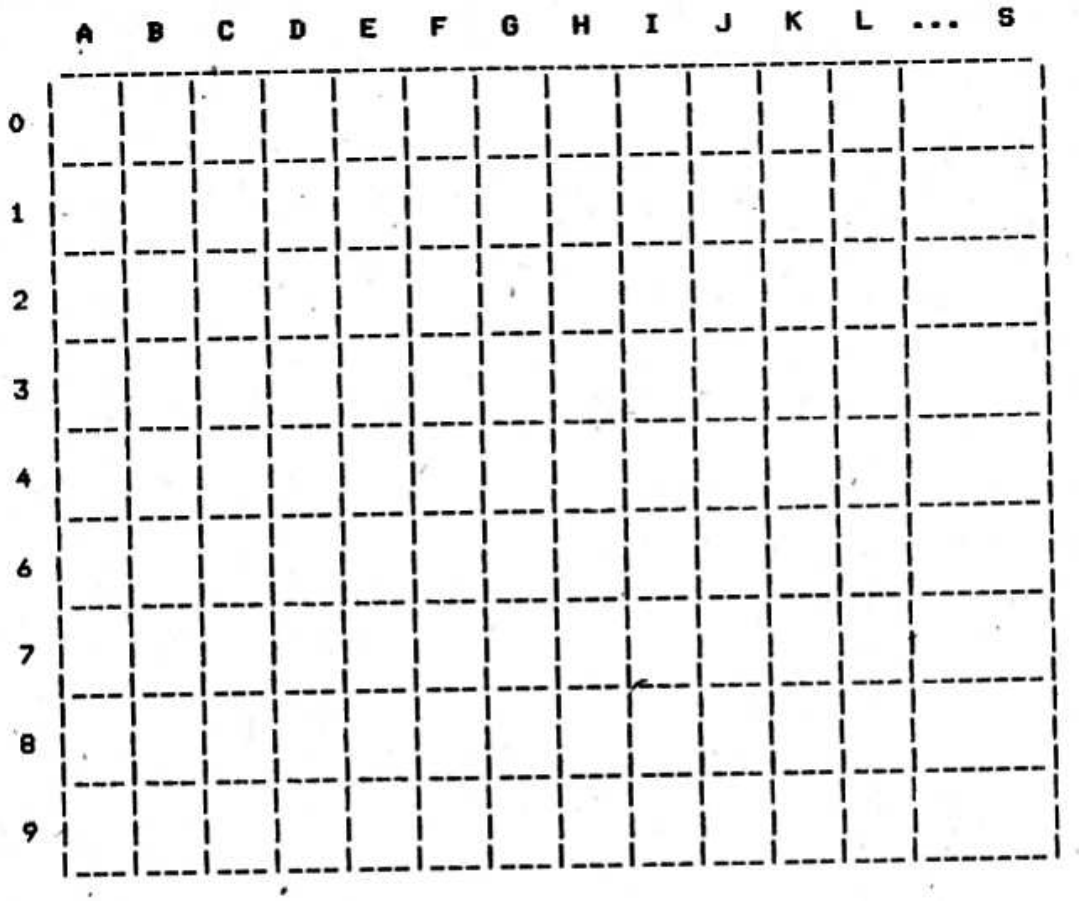
Beide werden solange in der Matrix die "Suche" stattfindet darunter, eingeblendet. Ist das Feld belegt, wird der Wert angezeigt, sonst bleibt das "*"-Zeichen im Feldelement stehen.
Anschliessend wird erneut zur Eingabe von Horizontal- und Vertikalposition aufgefordert. Dies erfolgt bis zum Erreichen der vorgegebenen Rundenzahl.
Links unterhalb der Matrix gibt das System die zur Zeit aktuelle Spielrunde aus.

- Zum Schluss wird die Treffersumme angezeigt und mit dem Einblenden der Ausschrift

```
** Spiel-Ende **
```

das Spiel abgeschlossen.

- **Aufbau Bildschirmbild**



- **Quellprogrammliste**

```
1000 '      T R E F F E R  -  S P I E L
1010 '      ===========================
1020 '
1022 'Definition Kursorsteuerfunktion
1023 '
1030 DEF FNC$(Y,X)=CHR$(27)+CHR$(Y+127)+CHR$(X+127)
1032 '
1033 'Vereinbarung Matrixfeld
1034 '
1040 DIM G(20,20)
1050 '
1060 '
1070 ' Initialisierung Spiel :
1080 '      Eingabe Steuerzahl zur Belegung der Spielmatrix
1090 '
```

```
1100 INPUT "Eingabe Zahl  im Bereich von 1 ... 5000 : ",A%
1101 '
1102 'Anfangszahl Zufallszahlengenerator
1103 'einstellen1104'
1110 RANDOMIZE A%
1120 '
1140 '
1150 R%=0                              ' R% = akt. Spielrundenzahl
1160 INPUT "Spielrundenzahl : ",A%  ' A% = vorgeg. Spielrundenz.
1170 U% = 0                           ' U% = akt. Zahl bel. Matrixel.
1180 '
1190 '
1200 ' Anzeige Spielfeld-Matrix
1210 '
1220 PRINT CHR$(12) CHR$(13) " ";
1230 FOR C%=65 TO 83:PRINT"   " CHR$(C%);:NEXT:PRINT
1240 FOR L%=0 TO 9
1250   PRINT"  |";:FOR C%=1 TO 19:PRINT"---|";:NEXT:PRINT
1260   PRINT USING "# |";L%;:FOR C%=1 TO 19:PRINT"   |";:
       NEXT:PRINT
1270 NEXT:PRINT"  |";:FOR C%=1 TO 19:PRINT"---|";:NEXT
1280 '
1290 '
1300 ' Aufruf zur Belegung der Spielfeld-Matrix mit
1310 ' Werten (unsichtbar fuer den Spieler)
1320 '
1330 ' K=Kennzahl
1340 ' S=Zusammenhaengende Kennzahl gleichen Wertes
1350 ' N=Anzahl
1360 '
1370 K=2:N=1:S=4:GOSUB 1920
1380 K=3:N=2:S=4:GOSUB 1920
1390 K=4:N=3:S=3:GOSUB 1920
1400 K=5:N=4:S=2:GOSUB 1920
1410 K=6:N=5:S=2:GOSUB 1920
1420 K=7:N=6:S=1:GOSUB 1920
1440 '
1450 '
1460 '
1470 '       Spiel - Durchfuehrung
1480 '
1490 ' Ausgabe Spielrundennummer
1500 ' Eingabe vermuteter Treffer  in Matrix
1510 ' durch Eingabe Horizontal- und Vertikalposition
1520 '
1530 R%=R%+1
1540 PRINT CHR$(13) FNC$(23,1) CHR$(20)
     USING "  Runde ### von ###";R%;A%
1550 PRINT FNC$(24,10) "Eingabe Horizontal (A-S)  : ";
1560 H$=INPUT$(1):IF H$<"A" OR H$>"S" THEN 1560
1570 PRINT H$ "  Vertikal (0-9) : ";
1580 V$=INPUT$(1):IF V$<"0" OR V$>"?" THEN 1580
     ELSE IF V$>"9" THEN 1540
1590 PRINT V$ CHR$(13) FNC$(23,63) "Start   ";
1600 FOR D%=10 TO 0 STEP -1
1610   PRINT USING "##";D%;:PRINT STRING$(2,8);
```

```
1620   FOR C%=1 TO 200:NEXT
1630 NEXT
1660 '
1670 ' Suchvorgang in Matrix
1680 ' --> ist auf vermuteten Matrixel. Treffer vorhanden ?
1690 '
1700 ' --> wenn Treffer erzielt, wird Trefferwert angezeigt
1710 '
1720 ' --> wenn kein Treffer erzielt wird, so wird * in
1730 '     Matrixelement vermerkt
1740 '
1750 '
1760 PRINT CHR$(13) FNC$(23,35) "*** Suche **":
     FOR C%=0 TO 1000:NEXT
1770 X=ASC(H$)-65:Y=VAL(V$):
     PRINT FNC$(3+Y*2,5+X*4-1) " * ":G(X,Y+10)=1
1780 FOR C%=1 TO 3000:NEXT
1790 IF G(X,Y)=0 THEN 1850
1800 T%=G(X,Y):RESTORE 1900
1810 FOR P%=1 TO T%:READ T$:NEXT
1820 PRINT CHR$(13) FNC$(3+Y*2,5+X*4-1) T$
1840 U%=U%-1:
1850 IF U%>0 AND A%>R% THEN 1450
1860 PRINT CHR$(13) FNC$(23,1) CHR$(20);
1870 PRINT FNC$(23,30) "*** Spiel-Ende ***" FNC$(22,1);
1880 GOSUB 2500
1890 END
1900 DATA " * "," 2 "," 3 "," 4 "," 5 "," 6 "," 7 "," 8 "
1910 '
1920 '
1930 '  Belegung der Spielfeld-Matrix mit Werten
1940 '
1950 '   ---------------------------------------
1960 '   | Matrix           | Matrixelement mit  |
1970 '   |eingestellt       | belegt, wo Spie-   |
1980 '   |                  | ler Zahl vermutet  |
1990 '   |                  |                    |
2000 '   |                  |                    |
2010 '   ---------------------------------------
2020 '
2030 '
2040 U%=U%+N*S:
2050 FOR C%=1 TO N
2052 '
2053 '                    Auswahl Startelement (zufallsbedingt)
2054 '
2060   X=INT(RND(1)*19):Y=INT(RND(1)*10):
       IF G(X,Y)<>0 THEN 2060 ELSE G(X,Y)=K
2070   IF S=1  THEN 2420
2071 '
2072 '                    AUSWAHL KETTENEINTRAGUNGSRICHTUNG
2073 '                    (zufallsbedingt)
2074 '
2080   W%=0:I%=INT(RND(1)*5+1):
       ON I% GOTO 2100,2120,2180,2240,2300,2360
2090 '
```

```
2100   G(X,Y)=0:GOTO 2060
2110 '
2111 '                                          Waagerechte Kette
2112 '
2120   IF X+S-1>18 THEN 2080
2130   FOR Z%=1 TO S-1:IF G(X+Z%,Y)=0 THEN W%=W%+1
2140   NEXT
2150   IF W%<>S-1 THEN 2080
2160   FOR Z%=1 TO S-1:G(X+Z%,Y)=K:NEXT:GOTO 2420
2170 '
2171 '                                          Senkrechte Kette
2172 '
2180   IF Y+S-1>9 THEN 2080
2190   FOR Z%=1 TO S-1:IF G(X,Y+Z%)=0 THEN W%=W%+1
2200   NEXT
2210   IF W%<>S-1 THEN 2080
2220   FOR Z%=1 TO S-1:G(X,Y+Z%)=K:NEXT:GOTO 2420
2230 '
2231 '                        Diagonale nach rechts unten
2232 '
2240   IF X+S-1>18 OR Y+S-1>9 THEN 2080
2250   FOR Z%=1 TO S-1:IF G(X+Z%,Y+Z%)=0 THEN W%=W%+1
2260   NEXT
2270   IF W%<>S-1 THEN 2080
2280   FOR Z%=1 TO S-1:G(X+Z%,Y+Z%)=K:NEXT:GOTO 2420
2289 '
2290 '                        Diagonale nach rechts oben
2291 '
2300   IF X+S-1>18 OR Y-S-1<0 THEN 2080
2310   FOR Z%=1 TO S-1:IF G(X+Z%,Y-Z%)=0 THEN W%=W%+1
2320   NEXT
2330   IF W%<>S-1 THEN 2080
2340   FOR Z%=1 TO S-1:G(X+Z%,Y-Z%)=K:NEXT:GOTO 2420
2350 '
2351 '                        Diagonale nach links unten
2352 '
2360   IF X-S-1<0 OR Y+S-1>9 THEN 2080
2370   FOR Z%=1 TO S-1:IF G(X-Z%,Y+Z%)=0 THEN W%=W%+1
2380   NEXT
2390   IF W%<>S-1 THEN 2080
2400   FOR Z%=1 TO S-1:G(X-Z%,Y+Z%)=K:NEXT:GOTO 2420
2410 '
2420 NEXT
2430 RETURN
2440 '
2450 '
2460 ' Auswertung :
2470 '
2480 '   Ausgabe der erzielten Treffer (Summe)
2490 '
2500 SUM=0
2510 FOR V%=0 TO 9:FOR H%=0 TO 18
2520   IF G(H%,V%+10)=1 THEN 2560
       ELSE PRINT CHR$(13) FNC$(3+V%*2,5+H%*4-1);
2530 NEXT:PRINT:NEXT
2540 PRINT:PRINT "Treffersumme : ";SUM
```

```
2550 RETURN
2560 SUM=SUM+G(H%,V%)
2570 GOTO 2530
```

2.4.2. KALENDER

- Kurzcharakteristik des Programmes

Zur Demonstration des KALENDER-Programmes ist eine Konfiguration mit Drucker erforderlich. Das KALENDER-Programm meldet sich mit der Bildschirmanzeige:

```
Eingabe Jahr:
```

Die Jahreszahl fuer den zu druckenden Kalender ist einzugeben (z.B. 1985).

Das Programm gestattet die Ausgabe fuer die Kalenderjahre 1582 bis 4882 (Gregorianischer Kalender).

Programmintern wird eine Datumsmatrix aufgebaut. Bei ihrer Berechnung wird das Schaltjahr beruecksichtigt.
Dabei gilt:

- Ein Jahr ist dann ein Schaltjahr, wenn es durch 4 teilbar ist und kein volles Jahrhundert vorliegt.

- Ein Jahr ist dann ein Schaltjahr, wenn es ein Vielfaches von 400 ist.

- Weitere BASIC-Ausdrucksmittel, die am Beispiel "TREFFER" noch nicht betrachtet wurden:

- Programmierung von Druckmasken auf die durch Unterprogramme wiederholt zugegriffen wird:

```
450 GOSUB 570

530 IF I/6=FIX(I/6) THEN GOSUB 562

562 LPRINT "=========================   ";
563 LPRINT "=========================   ";
564 LPRINT "========================="
565 RETURN
```

Der Sprung erfolgt dabei in Abhaengigkeit vom Wert durch ON ... GOSUB.

```
540 IF I/6=FIX(I/6) THEN IF FIX(I/6)<>4 THEN ON I/6
    GOSUB 640,680,720
```

Mittels READ - DATA wird den einzelnen Monaten eine unterschiedliche Tageszahl zugeordnet:
z. B. Maerz = 31 Tage.

```
250 READ A(1),A(2),A(3),A(4),A(5),A(6),A(7),A(8),
    A(9),A(10),A(11),A(12)
260 DATA 31,28,31,30,31,30,31,31,30,31,30,31
```

Quellprogrammliste

```
5 REM            KALENDER
6 '
7 '
8 '
9 REM        Eingabe Kalenderjahreszahl
10 INPUT "Eingabe  Jahr : ",K
11 '
12 '
15 REM       Druck Kalenderjahreszahl
20 LPRINT TAB(37);USING"#####";K
22 LPRINT TAB(37);" ===="
23 LPRINT:LPRINT
24 '
25 '
27 REM       Aufbau einer Datumsmatrix:
             Spalte:  3 Monate mit 7 Tagesspalten
             Zeile:   4 Monate mit 6 Tageszeilen
28 '
29 '         Vereinbarung der Matrixfelder
30 DIM A(12)
40 N=K
50 C=7
60 DIM B(24,21)
70 X=3
71 '
72 '
73 '         Berechnungsgrundlage:  1980
80 IF N)=1980 THEN GOTO 180
90 FOR I=1980 TO N+1 STEP -1
100 IF (I-1)/4=FIX((I-1)/4) THEN X=I-1
110 IF (I-1)/100=FIX((I-1)/100) THEN X=X+I
120 IF (I-1)/400=FIX((I-1)/400) THEN X=X-I
130 X=X-1
140 IF X)7 THEN X=2
150 IF X(1 THEN X=7
160 NEXT I
170 GOTO 250
180 IF N=1980 THEN GOTO 250
190 FOR I=1980 TO N-1
200 IF I/4=FIX(I/4) OR I/400=FIX(I/400) THEN X=X+1
210 IF I/100=FIX(I/100) THEN X=X-1
220 X=X+1
230 IF X)7 THEN X=1
240 NEXT I
241 '
```

```
242 '
243 REM  Tageszahl / Monat einlesen aus DATA-Anweisung.
       Schaltjahr wird beruecksichtigt, wenn
       erforderlich.
250 READ A(1),A(2),A(3),A(4),A(5),A(6),A(7),A(8),A(9),
       A(10),A(11),A(12)
260 DATA 31,28,31,30,31,30,31,31,30,31,30,31
270 IF N/4=FIX(N/4) OR N/400=FIX(N/400) THEN A(2)=29
280 IF N/100=FIX(N/100) THEN A(2)=28
281 '
282 '
283 REM   Zuordnung der Tage in die Datumsmatrix
290 FOR I=1 TO 12
300 Y=1
310 IF I)=4 THEN Y=7
320 IF I)=7 THEN Y=13
330 IF I)=10 THEN Y=19
340 FOR J=1 TO A(I)
350 B(Y,X)=J
360 X=X+1
370 IF X)C THEN Y=Y+1
380 IF X)C THEN X=C-6
390 NEXT J
400 C=C+7
410 X=X+7
420 IF X)=22 THEN C=7
430 IF X)=22 THEN X=X-21
440 NEXT I
441 '
442 '
443 '
444 REM      AUSDRUCK KALENDER
445 '
446 '        Druck des Monatskopfes (Januar - Maerz,
                                     Sonntag - Sonnabend)
450 GOSUB 570
451 '
452 '
453 '        Zaehler fuer Zeile
460 FOR I=1 TO 24
462 '
463 '        Zaehler fuer Spalte
470 FOR J=1 TO 21
471 '
472 '
473 '        Druck des Rahmens
480 IF J=1 THEN LPRINT "!";
490 IF(J-1)/7=FIX((J-1)/7) THEN IF J<>1 THEN
    LPRINT " !   !";
491 '
492 '
493 '        Druck Datum
500 IF B(I,J)>0 THEN LPRINT USING "###";B(I,J);
    ELSE LPRINT "   ";
510 NEXT J
520 LPRINT " !"
```

```
521 '
522 '
523 '        Druck Monatsabschluss sowie Sprungverteiler
            zum Druck der restlichen Monate
530 IF I/6=FIX(I/6) THEN GOSUB 562
540 IF I/6=FIX(I/6) THEN IF FIX(I/6)()4 THEN ON I/6
    GOSUB 640,680,720
550 NEXT I
551 '       Bildschirm loeschen / Programmende
552 LPRINT CHR$(12)
560 END
561 '       DRUCKMASKEN
562 LPRINT "=========================    ";
563 LPRINT "=========================    ";
564 LPRINT "========================="
565 RETURN
570 LPRINT
580 LPRINT
590 LPRINT
    "           JANUAR                          FEBRUAR          ";
591 LPRINT "            MAERZ"
600 LPRINT
    "!     ------------     !   !     ------------  !   ";
601 LPRINT "!     ------------     !"
610 LPRINT
    "!  S  M  D  M  D  F  S !   ! S  M  D  M  D  F  S !";
611 LPRINT  "!  S  M  D  M  D  F  S !"
620 LPRINT
    "!----------------------!   !--------------------!";
621 LPRINT "!----------------------!"
630 RETURN
640 LPRINT
650 LPRINT
660 LPRINT
"          APRIL                           MAI              ";
661 LPRINT "        JUNI"
670 GOTO 600
680 LPRINT
690 LPRINT
700 LPRINT
"          JULI                           AUGUST           ";
701 LPRINT "        SEPTEMBER"
710 GOTO 600
720 LPRINT
730 LPRINT
740 LPRINT
"        OKTOBER                         NOVEMBER         ";
741 LPRINT "        DEZEMBER"
750 GOTO 600
```

2.4.3. WOCHENTAG

- Kurzcharakteristik des Programmes

Das Programm ermittelt verschiedene Angaben zu einem bestimmten, vom Bediener eingegebenen Datum.

Auf dem Bildschirm werden angezeigt:

- Der ermittelte Wochentag
- Glueckwuensche zum Geburtstag, wenn das eingegebene Datum (Tag, Monat) mit dem Tagesdatum uebereinstimmt.
- Das ermittelte Alter (bei Eingabe Geburtsdatum)
- Die errechnete Zeit fuer:
  - Schlaf
  - Essen
  - Arbeit / Studium / Spiel
  - Ruhe.
- Die Jahreszahl, zu der das Rentenalter erreicht ist. Dabei werden 65 Jahre als Rentenalter angesetzt, und das Geburtsdatum ist einzugeben.

Es wird eine Datumsmatrix mit folgender Wertzuordnung zu den einzelnen Monaten aufgebaut:

Januar / Oktober:	0
Februar / Maerz / November:	3
April / Juli:	6
Mai:	1
Juni:	4
September / Dezember:	5
August:	2

Diese Werte kennzeichnen den Tag des Monatsbeginns.

In einer gesonderten Routine wird das Schaltjahr beruecksichtigt, wobei bei Vorliegen eines Schaltjahres diese Werte modifiziert werden.

- Bedienung

Folgende Programmeingaben sind notwendig:

- Es ist das Tagesdatum einzugeben.
- Es ist ein Datum einzugeben, fuer das die o.g. Angaben ermittelt werden sollen.
  Dabei ist zu beachten, dass eine Jahreszahl nach 1582 (Gregorianischer Kalender) einzugeben ist.

Fuer beide Eingaben ist die Form Monat, Tag, Jahr (z.B. 3,24,1984) zu waehlen.

- Weitere BASIC-Ausdrucksmittel, die bisher noch nicht naeher betrachtet wurden

- Es werden anwendereigene Funktionen definiert, die im Programm zur Berechnung verschiedener Variablen genutzt werden:

```
170 DEF FNA(A)=INT(A/4)

190 DEF FNB(A)=INT(A/7)
```

Auf diese Funktionen wird wiederholt im Programm zugegriffen.

Ausserdem wird im Programm "WOCHENTAG" wiederholt auf die arithmetische Funktion INT zurueckgegriffen, die den ganzzahligen Wert z.B. der Variablen B ermittelt.

```
310 LET I2=INT(A-FNB(A)*7)

350 LET B=INT(A-FNB(A)*7)+1

380 LET T1=INT(Y-FNA(Y)*4) usw.
```

- In Abhaengigkeit vom Ergebnis der Berechnung einer Variablen wird an unterschiedliche Stellen im Programm verzweigt. Dies wird zum Beispiel fuer die Auswahl des entsprechenden Wochentages genutzt.

```
490 IF (Y1*12+M1)*31+D1<(Y*12+M)*31+D THEN 550
500 IF (Y1*12+M1)*31+D1=(Y*12+M)*31+D THEN 530
510 PRINT M;"/";D;"/";Y;" WAR EIN ";
520 GOTO 570
530 PRINT M;"/";D;"/";Y;" IST EIN ";
540 GOTO 570
550 PRINT M;"/";D;"/";Y;" WIRD SEIN EIN ";
570 IF B <>1 THEN 590
580 PRINT "SONNTAG."
590 IF B<>2 THEN 610
600 PRINT "MONTAG."
610 IF B<>3 THEN 630
620 PRINT "DIENSTAG."
630 IF B<>4 THEN 650
640 PRINT "MITTWOCH."
650 IF B<>5 THEN 670
660 PRINT "DONNERSTAG."
670 IF B<>6 THEN 690
680 GOTO 1250
690 IF B<>7 THEN 710
700 PRINT "SONNABEND."
```

- Beispiele zur Angabe verschiedener Informationen zu einem speziell ausgewaehlten Datum:

```
                                W O C H E N T A G

W O C H E N T A G  IST EIN COMPUTERPROGRAMM, DASS
INFORMATIONEN UEBER EINEN BESTIMMTEN TAG VERMITTELT.

EINGABE HEUTIGES DATUM IN DER FORM: 3,24,1979 ?
EINGABE  GEBURTSDATUM  ODER ANDERES INTERESSANTES DATUM
(NACH 1582)

2 / 12 / 1940  WAR EIN MONTAG

***HERZLICHEN GLUECKWUNSCH***
                                JAHRE       MONATE      TAGE
                                -----       ------      ----
IHR ALTER (BEI GEBURTSTAG)       45           0          0
SIE HABEN GESCHLAFEN             15           9          3
SIE HABEN GEGESSEN                7           7         27
SIE HABEN GEARBEITET             10           4          7
SIE HABEN AUSGESPANNT            11           2         23

              ***  SIE WERDEN IN RENTE GEHEN  2005  ***
```

```
                                W O C H E N T A G

W O C H E N T A G  IST EIN COMPUTERPROGRAMM, DASS
INFORMATIONEN UEBER EINEN BESTIMMTEN TAG VERMITTELT.

EINGABE HEUTIGES DATUM IN DER FORM : 3,24,1979 ?
EINGABE GEBURTSDATUM ODER ANDERES INTERESSANTES DATUM
(NACH 1582)

1 / 13 / 1978 WAR EIN FREITAG DER DREIZENHTE ---BEWAHRE!
                                JAHRE       MONATE      TAGE
                                -----       ------      ----
IHR ALTER (BEI GEBURTSTAG)        7           0         29
SIE HABEN GESCHLAFEN              2           5         24
SIE HABEN GEGESSEN                1           2         14
SIE HABEN GESPIELT                1           7         19
SIE HABEN AUSGESPANNT             1           9          2

              ***  SIE WERDEN IN RENTE GEHEN 2043  ***
```

- Quellprogrammliste

```
1 REM            WOCHENTAG
2 F$="  ##             ##             ##"          ' Druckmaske
5 PRINT CHR$(12):WIDTH 80
10 PRINT TAB(32);"W O C H E N T A G":PRINT
30 PRINT:PRINT:PRINT
40 '
50 '
60 REM     Eingabe Tagesdatum sowie Datum, zu dem Angaben
           ermittelt werden sollen
70 '
80 '
100 PRINT "W O C H E N T A G   IST EIN COMPUTERPROGRAMM, DASS"
110 PRINT
   "INFORMATIONEN UEBER EINEN BESTIMMTEN TAG VERMITTELT."
120 PRINT
130 PRINT
   "EINGABE HEUTIGES DATUM IN DER FORM: 3,24,1979 ? "; TAB(68);
140 INPUT "",M1,D1,Y1
150 '
151 '
152 '
160  REM   Anwendereigene Funktionsdefinition und Vereinbarung
           Matrixfelder
170 DEF FNA(A)=INT(A/4)
180 DIM T(12)
190 DEF FNB(A)=INT(A/7)
200 '
201 '
202 '
203 REM   Monatswert fuer Datumsmatrix einlesen
210 FOR I= 1 TO 12
220 READ T(I)
230 NEXT I
231 '
232 '
240 PRINT
    "EINGABE GEBURTSDATUM ODER ANDERES INTERESSANTES DATUM ";
241 PRINT "(NACH 1582)";
250 INPUT M,D,Y
260 PRINT CHR$(12)
261 '
262 '
263 REM   Berechnung Wochentag
270 LET I1 = INT((Y-1500)/100)
280 '
281 REM   Abfrage, ob Datum vor 1582 eingegeben wurde
290 IF Y-1582 <0 THEN 1300
291 '
292 '
300 LET A = I1*5+(I1+3)/4
310 LET I2=INT(A-FNB(A)*7)
320 LET Y2=INT(Y/100)
330 LET Y3 =INT(Y-Y2*100)
340 LET A =Y3/4+Y3+D+T(M)+I2
```

```
350 LET B=INT(A-FNB(A)*7)+1
360 IF M > 2 THEN 470
370 IF Y3 = 0 THEN 440
380 LET T1=INT(Y-FNA(Y)*4)
390 IF T1 <> 0 THEN 470
400 IF B<>0 THEN 420
410 LET B=6
420 LET B = B-1
430 GOTO 470
440 LET A = I1-1
450 LET T1=INT(A-FNA(A)*4)
460 IF T1 = 0 THEN 400
470 IF B <>0 THEN 490
480 LET B = 7
481 '
482 '
483 REM   ANZEIGE   W O C H E N T A G
490 IF (Y1*12+M1)*31+D1<(Y*12+M)*31+D THEN 550
500 IF (Y1*12+M1)*31+D1=(Y*12+M)*31+D THEN 530
510 PRINT M;"/";D;"/";Y;" WAR EIN ";
520 GOTO 570
530 PRINT M;"/";D;"/";Y;" IST EIN ";
540 GOTO 570
550 PRINT M;"/";D;"/";Y;" WIRD SEIN EIN ";
570 IF B <>1 THEN 590
580 PRINT "SONNTAG."
590 IF B<>2 THEN 610
600 PRINT "MONTAG."
610 IF B<>3 THEN 630
620 PRINT "DIENSTAG."
630 IF B<>4 THEN 650
640 PRINT "MITTWOCH."
650 IF B<>5 THEN 670
660 PRINT "DONNERSTAG."
670 IF B<>6 THEN 690
680 GOTO 1250
690 IF B<>7 THEN 710
700 PRINT "SONNABEND."
710 IF (Y1*12+M1)*31+D1=(Y*12+M)*31+D THEN 1120
711 '
712 '
713 REM  Berechnung und Anzeige der aktuellen Angaben zum
         eingegebenen Datum
714 '
715 '
720 LET I5=Y1-Y
730 PRINT
740 LET I6=M1-M
750 LET I7=D1-D
760 IF I7>=0 THEN 790
770 LET I6= I6-1
780 LET I7=I7+30
790 IF I6>=0 THEN 820
800 LET I5=I5-1
810 LET I6=I6+12
820 IF I5<0 THEN 1310
```

```
830 IF I7 <> 0 THEN 850
835 IF I6 <> 0 THEN 850
836 '
837 '
838 '
839 REM     ANZEIGE   ermittelte Jahre / Monate / Tage
840 PRINT"***HERZLICHEN GLUECKWUNSCH***"
850 PRINT " ","  ","JAHRE","MONATE","TAGE"
855 PRINT " ","  ","-----","------","----"
860 PRINT
    "IHR ALTER (BEI GEBURTSTAG) ",:PRINT USING F$;I5,I6,I7
861 '
862 '
870 LET A8 = (I5*365)+(I6*30)+I7+INT(I6/2)
880 LET K5 = I5
890 LET K6 = I6
900 LET K7 = I7
920 LET E = Y+65
940 LET F = .35
941 '
942 '
950 PRINT "SIE HABEN GESCHLAFEN ",
960 GOSUB 1370                  ' Sprung zum Anzeige-Unterprogramm
970 LET F = .17
980 PRINT "SIE HABEN GEGESSEN ",
990 GOSUB 1370
1000 LET F = .23
1010 IF K5 > 3 THEN 1040
1020 PRINT "SIE HABEN GESPIELT",
1030 GOTO 1080
1040 IF K5 > 9 THEN 1070
1050 PRINT "SIE HABEN STUDIERT ",
1060 GOTO  1080
1070 PRINT "SIE HABEN GEARBEITET",
1080 GOSUB 1370
1085 GOTO 1530
1086 '
1087 '
1090 PRINT "SIE HABEN AUSGESPANNT ",:PRINT USING F$;K5,K6,K7
1100 PRINT
1110 PRINT TAB(16);"***  SIE WERDEN IN RENTE GEHEN ";E;" ***"
1120 PRINT
1140 PRINT
1200 END
1201 '
1202 '
1203 REM     Angaben zum Freitag
1210 PRINT
1220 PRINT
1230 PRINT
1250 IF D=13 THEN 1280
1260 PRINT "FREITAG."
1270 GOTO 710
1280 PRINT "FREITAG DER DREIZEHNTE---BEWAHRE!"
1290 GOTO 710
1291 '
```

```
1292 '
1293 REM     Fehlerhafte Eingabe
1300 PRINT "ES WURDE EIN TAG VOR  MDLXXXII  EINGEGEBEN. "
1310 GOTO 1140
1320 '
1321 '
1322 '        Werte fuer Datumsmatrix
1330 DATA 0, 3, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5
1340 '
1350 '
1360 REM   Unterprogramm zur Anzeige der Jahres-, Monats- und
           Tageszahl
1370 LET K1=INT(F*A8)
1380 LET I5 = INT(K1/365)
1390 LET K1 = K1- (I5*365)
1400 LET I6 = INT(K1/30)
1410 LET I7 = K1 -(I6*30)
1420 LET K5 = K5-I5
1430 LET K6 =K6-I6
1440 LET K7 = K7-I7
1450 IF K7)=0 THEN 1480
1460 LET K7=K7+30
1470 LET K6=K6-1
1480 IF K6)0 THEN 1510
1490 LET K6=K6+12
1500 LET K5=K5-1
1510 PRINT USING F×;I5,I6,I7
1520 RETURN
1521 '
1522 '
1523 REM   Ermitteln der Werte fuer letzte Zeile der Anzeige
1530 IF K6=12 THEN 1550
1540 GOTO 1090
1550 LET K5=K5+1
1560 LET K6=0
1570 GOTO 1090
```

1985

JANUAR

S	M	D	M	D	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

FEBRUAR

S	M	D	M	D	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28		

MAERZ

S	M	D	M	D	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

APRIL

S	M	D	M	D	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

MAI

S	M	D	M	D	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

JUNI

S	M	D	M	D	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

JULI

S	M	D	M	D	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

AUGUST

S	M	D	M	D	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

SEPTEMBER

S	M	D	M	D	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

OKTOBER

S	M	D	M	D	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

NOVEMBER

S	M	D	M	D	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

DEZEMBER

S	M	D	M	D	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

3. Zusatzpaket 1/2"-Magnetbandunterstuetzung im BASIC-Compiler BASICMB 1520 (SCPX)

Zur Nutzung 1/2"-Magnetband CM 5300.01 des A5120/30 fuer die Erstellung von anwenderspezifischer Konvertier- oder Archivierungssoftware wird ein Zusatzpaket fuer den BASIC-Compiler BASC 1520 (SCPX) bereitgestellt. Es besteht aus den Komponenten OBSLIB.REL, DUM.REL, XX.COM und MBTEST.COM. Eine analoge Unterstuetzung unter dem BASIC-Interpreter ist nicht moeglich.

3.1. Technologie

3.1.1. Compilieren BASIC-Programm

```
A> BASC (relfile),=(sourcefile)/O
```

Mittels dieses Kommandos erfolgt das Uebersetzen des als ASCII-Datei gespeicherten BASIC-Programms in ein Objektprogramm. Durch den Schalter O werden Objektprogrammanweisungen zur Verarbeitung der Bibliothek OBSLIB.REL in das Zielprogramm eingefuegt.

3.1.2. Verbinden Objektprogramm

```
A> LINK /P:300,OBSLIB/S,(relfile),DUM,(comfile)/N/E
```

Das BASIC-Programm wird ab Adresse 300H gebunden, da zuvor die MB-Steuerung angeordnet werden muss.
Das Einlagern der MB-Steuerung erfolgt nach dem Linken, da erst nach dem Start des BASIC-Programms die Verschiebung auf die Adresse 300H erfolgt. Globale Referenzen zwischen BASIC-Programm und MB-Steuerung werden durch den Modul DUM beruecksichtigt.

Die nach dem Binden ausgegebene Startadresse des BASIC-Programms sowie die Angabe der Zahl der 256-Byte-Bloecke, die durch das Programm belegt werden, werden fuer die Ausgabe des aufbereiteten Programms (siehe 3.1.4.) benoetigt.

3.1.3. Ueberlagerung mit MB-Steuerroutine

Die MB-Steuerroutine besteht aus dem MB-Modul und der Parametervermittlungsroutine (PVP), beide zusammen stehen als Datei XX.COM bereit.

Die Ueberlagerung erfolgt mittels des Debuggers DU in folgenden Schritten:

```
A> DU
#I(comfile)                    ) Verschiebung
#R                             ) Objektprogramm
#G100,117                      )

#IXX.COM                       ) Ueberlagerung
#R                             ) MB-Steuerroutine
#S100
0100 |_| C3                    ) Nachtragen Sprungbefehl
0101 |_| __  L-Teil            ) mit Startadresse des
0102 |_| __  H-Teil            ) BASIC-Programmes
```

3.1.4. Ausgabe BASIC-Programm auf Diskette

Das BASIC-Programm einschliesslich MB-Steuerung ist mittels des Kommandos SAVE auf Diskette auszugeben:

```
A> SAVE S (comfile)
```

Dabei ist S die Zahl der 256-Byte-Bloecke des BASIC-Programmes plus 2 Bloecke.
Uber den comfile-Namen ist das BASIC-Programm wie jedes andere abarbeitbare Programm startbar.

3.2. Aufruf Magnetbandroutine in BASIC

Die Magnetbandroutine wird ueber die Anweisung

```
CALL MB(P1%,P2%,P3%)
```

aufgerufen.

Fuer den Aufruf sind den 3 Parameter die Werte

- P1% - Blocklaenge
- P2% - Blockadresse
- P3% - Funktionscode

zu uebergeben.

Fuer die Wertrueckgabe aus dem MB-Modul werden die Parameter mit

- P1% - Ergebnisvektor
- P2% - Blockadresse
- P3% - Statusbyte

belegt.

Das Einstellen der Parameter erfolgt gemaess der Beschreibung des MB-Treibers in der "Zusatzsoftware II".
Die Rueckmeldungen des Treibers (die Belegung des Statusbytes) ist ebenfalls der Beschreibung des MB-Treibers in der "Zusatzsoftware II" zu entnehmen.

3.3. Fallbeispiel

Als Anlage 1 wird ein BASIC-Demonstrationsprogramm zur Arbeit mit 1/2"-MB angegeben.

Anlage 2 zeigt die Abarbeitung dieses BASIC-Programms. Der Dialog zur Realisierung einiger Funktionen zur MB-Arbeit ist protokolliert.

3.4. Dateien

OBSLIB.REL	- BASIC-Standardbibliothek
DUM.REL	- Pseudo-Modul zur Sicherung globaler Referenzen
XX.COM	- MB-Steuerroutine (MB-Modul und Parametervermittlungsroutine)
MBTEST.COM	- BASIC-Demonstrationsprogramm einschliesslich MB-Steuerroutine
BASC.COM	- BASIC-Compiler
LINK.COM	- Programmverbinder

Anlage 1

BASIC-Demonstrationsprogramm

```
5 PRINT
10 PRINT "Funktionsauswahl"
15 PRINT "================"
20 PRINT
30 PRINT "Schreiben" TAB(30) "1"
40 PRINT "Lesen" TAB(30) "2"
50 PRINT "BM schreiben" TAB(30) "3"
60 PRINT "Block vorsetzen" TAB(30) "4"
70 PRINT "Block ruecksetzen" TAB(30) "5"
80 PRINT "Datei (BM) vorsetzen" TAB(30) "6"
90 PRINT "Datei (BM) ruecksetzen" TAB(30) "7"
100 PRINT "Rueckspulen" TAB(30) "8"
110 PRINT
120 INPUT "Funktionseingabe (Kennzahl) = ",K%
130 IF K%<1 OR K%>8 THEN 120
140 ON K% GOTO 1000,2000,3000,4000,5000,6000,7000,8000
150 '
160 '
170 I%=VARPTR(PUFFER$)
180 BA%=PEEK(I%+2)*256+PEEK(I%+1)
190 BL%=LEN(PUFFER$)
200 FCR%=FC%
210 CALL MB(BL%,BA%,FC%)
220 '
225 '
230 PRINT:PRINT
240 PRINT "Statusbyte = ";HEX$(FC%)
250 '
260 IF FCR%=2 THEN PRINT "Gelesener Datensatz :":
                   PRINT PUFFER$:PRINT
270 IF FCR%=2 OR FCR%=8 OR FCR%=136 OR FCR%=16 OR FCR%=144
                   THEN PRINT "Ergebnisvektor = ";BL%
280 PRINT:PRINT
290 INPUT "Programmende ? (J/N)  : ",X$
300 IF X$="J" OR X$="j" THEN END ELSE GOTO 5
310 '
320 '
330 '
340 '
1000 FC%=1
1010 INPUT "Ausgabedatensatz : ",PUFFER$
1020 GOTO 170
1030 '
1040 '
2000 FC%=2
2010 PUFFER$=SPACE$(255)
2020 GOTO 170
2030 '
2040 '
3000 FC%=4
3010 GOTO 210
```

```
3020 '
3030 '
4000 FC%=8
4010 INPUT "Anzahl Bloecke vorsetzen : ",BL%
4020 GOTO 210
4030 '
4040 '
5000 FC%=136
5010 INPUT "Anzahl Bloecke ruecksetzen :",BL%
5020 GOTO 210
5030 '
5040 '
6000 FC%=16
6010 INPUT "Anzahl Dateien (BM) vorsetzen : ",BL%
6020 GOTO 210
6030 '
6040 '
7000 FC%=144
7010 INPUT "Anzahl Dateien (BM) ruecksetzen : ",BL%
7020 GOTO 210
7030 '
7040 '
8000 FC%=32
8010 GOTO 210
8020 '
8030 '
8040 '*****************************************************
```

Anlage_2

```
Funktionsauswahl
================

Schreiben                 1
Lesen                     2
BM schreiben              3
Block vorsetzen           4
Block rüecksetzen         5
Datei (BM) vorsetzen      6
Datei (BM) ruecksetzen    7
Rueckspulen               8

Funktionseingabe (Kennzahl) = 1
Ausgabedatensatz : 1234567891111111111

Statusbyte = 0


Programmende ? (J/N) : n


Funktionsauswahl
================

Schreiben                 1
Lesen                     2
BM schreiben              3
Block vorsetzen           4
Block ruecksetzen         5
Datei (BM) vorsetzen      6
Datei (BM) rucksetzen     7
Rueckspulen               8

Funktionseingabe (Kennzahl) = 1
Ausgabedatensatz : 123456789222222222222222

Statusbyte = 0

Programmende ? (J/N) : n


Funktionsauswahl
================

Schreiben                 1
Lesen                     2
BM schreiben              3
Block vorsetzen           4
Block ruecksetzen         5
Datei (BM) vorsetzen      6
Datei (BM) ruecksetzen    7
Rueckspulen               8
```

```
Funktionseingabe (Kennzahl) = 3

Statusbyte = 20

Programmende ? (J/N) : n.


Funktionsauswahl
================

Schreiben                 1
Lesen                     2
BM schreiben              3
Block vorsetzen           4
Block ruecksetzen         5
Datei (BM) vorsetzen      6
Datei (BM) ruecksetzen    7
Rueckspulen               8

Funktionseingabe (Kennzahl) = 8

Statusbyte = 8

Programmende ? (J/N) : n


Funktionsauswahl
================

Schreiben                 1
Lesen                     2
BM schreiben              3
Block vorsetzen           4
Block ruecksetzen         5
Datei (BM) vorsetzen      6
Datei (BM) ruecksetzen    7
Rueckspulen               8

Funktionseingabe (Kennzahl) = 2

Statusbyte = 4
Gelesener Datensatz :
1234567891111111111

Ergebnisvektor = 19

Programmende ? (J/N) : n
```

```
Funktionsauswahl
================

Schreiben                1
Lesen                    2
BM schreiben             3
Block vorsetzen          4
Block ruecksetzen        5
Datei (BM) vorsetzen     6
Datei (BM) ruecksetzen   7
Rueckspulen              8

Funktionseingabe (Kennzahl) = 2

Statusbyte = 4
Gelesener Datensatz :
12345678922222222222222

Ergebnisvektor = 24

Programmende ? (J/N) : n


Funktionsauswahl
================

Schreiben                1
Lesen                    2
BM schreiben             3
Block vorsetzen          4
Block ruecksetzen        5
Datei (BM) vorsetzen     6
Datei (BM) ruecksetzen   7
Rueckspulen              8

Funktionseingabe (Kennzahl) = 2

Statusbyte = 20
Gelesener Datensatz :
```

4. Ergaenzungen zum FORTRAN- und C-Compiler

4.1. Ergaenzungen von FORTRAN-Compiler-Anweisungen

Anmerkung

Die Dokumention ersetzt den bisherigen Abschnitt 6.9. der Anwenderdokumentation "FORTRAN-Compiler unter dem Betriebssystem SCPX 1526". Weiterhin wird die Dokumentation um den Gliederungspunkt 6.10. erweitert. Diese Dokumentation wird bis zur Bereitstellung der ueberarbeiteten Auflage der FORTRAN-Dokumentation in der "Zusatzsoftware III" aufgefuehrt.

6.9. Einbinden von Assembler-Unterprogrammen

1. UP-Aufruf ohne Parameteruebergabe

Ein Unterprogrammaufruf ohne Parameter wird ueber einen einfachen CALL realisiert. Das entsprechende Unterprogramm wird ueber RET verlassen.
Das Unterprogramm wird mit dem Programm LINK (siehe "Anleitung fuer den Programmierer") mit dem FORTRAN-Programm gebunden.
Die Form des Aufrufs ist:

```
CALL name
```

wobei name eine INTEGER-Variable ist und den Eintrittspunkt in das Unterprogramm bezeichnet. Die Adressverbindung wird zum Zeitpunkt des Bindens hergestellt.

Beispiel:

Partielles Rollen des Bildschirms
Das Unterprogramm wird mit TP oder EDIT erfasst und damit eine .MAC-Datei erzeugt. Mit Hilfe des Assemblers ASM wird das Unterprogramm uebersetzt und eine .REL-Datei erstellt.

Das Unterprogramm UP.MAC lautet:

```
      .z80
      public up
up:
par1  equ  0f850h     ; 1. Zeile zum Rollen Bildschirm
par2  equ  0f800h     ; Anfangsadresse Bildschirm
par3  equ  5a0h       ; Laenge: 80 Zeichen, 18 Zeilen
      ld   hl,par1
      ld   de,par2
      ld   bc,par3
      ldir
      ret
      end.
```

Das FORTRAN-Programm wird mit TP erfasst und mit dem Compiler FOR 1520 (SCPX) uebersetzt in eine .REL-Datei.

Das FORTRAN-Programm UPOPAR.FOR lautet:

```
      INTEGER UP
      BYTE FF,A,B
      FF=12
      WRITE (1,100)FF
100   FORMAT(' ',A1)
      A=19
      B=1
      CALL KURPOS(A,B)
      WRITE (1,120)
120   FORMAT(' ','==========================================
     1==========================')
      A=21
      B=1
      CALL KURPOS(A,B)
      WRITE (1,130)
130   FORMAT(' ','FREIER TEIL ZUR BEDIENERFUEHRUNG')
200   CALL UP
      A=18
      B=1
      CALL KURPOS(A,B)
      WRITE (1,140)
140   FORMAT(' ',79X)
      A=19
      B=1
      CALL KURPOS(A,B)
      READ(1,150)K,L,M
150   FORMAT(3F10,3)
      GOTO 200
      END
      SUBROUTINE KURPOS(I,K)
      BYTE I,K,ESC,X,Y
      ESC=27
      X=I+127
      Y=K+127
      WRITE (1,110)ESC,X,Y
110   FORMAT(' ',3A1)
      RETURN
      END
```

Anschliessend werden die Programme UPOPAR.REL und UP.REL mit Hilfe des Programmmes LINK zu einem abarbeitungsfaehigen Programm des Typs .COM gebunden.

2. UP-Aufruf mit Parameteruebergabe

Ein Unterprogrammaufruf mit Parametern ergibt eine komplexere Aufruffolge. Parameter werden immer durch ihre Adresse uebergeben (Adresse des unteren Bytes des aktuellen Arguments). Es werden also unabhaengig vom Typ immer zwei Bytes belegt.

Die Art der Uebergabe ist abhaengig von der Anzahl der Parameter:

- Bis zu drei Parameter werden ueber Register uebertragen:

  Adresse des Parameters 1 in HL
  Adresse des Parameters 2 in DE
  Adresse des Parameters 3 in BC

- Ist die Anzahl der Parameter groesser als 3, steht im Register BC eine Adresse auf einen Datenblock, der die restlichen Parameter enthaelt.
  Das Unterprogramm muss die im Parameterblock (Datenblock) enthaltenen Parameter selbstaendig uebernehmen, d.h. die Anzahl der zu uebertragenen Parameter muss im Unterprogramm bekannt sein.
  Die Adresse des Parameters 1 steht immer in HL und des Parameters 2 in DE.
  Das aufrufende Programm ist verantwortlich fuer die richtige Uebermittlung der Parameter. Der Compiler nimmt keine Kontrolle vor.

Aus Gruenden der Einheitlichkeit der Uebergabe eines oder mehrerer Parameter ist es ratsam, diese Parameter mit Hilfe einer Parameterliste (Feld) zu uebertragen.

Beispiel: wie unter 1.

Uebergabe von 3 Parametern
Das Unterprogramm UP.MAC lautet:

```
        .z80
        public up
up:
        ld    a,m
        ld    (par1),a
        inc   hl
        ld    a,m
        ld    (par1+1),a
        inc   hl
;
        ld    a,m
        ld    (par2),a
        inc   hl
        ld    a,m
        ld    (par2+1),a
        inc   hl
;
        ld    a,m
        ld    (par3),a
        inc   hl
        ld    a,m
        ld    (par3+1),a
;
```

```
        ld    hl,(par1)
        ld    de,(par2)
        ld    bc,(par3)
;
        ldir
        ret
par1:   ds    2
par2:   ds    2
par3:   ds    2
        end
```

Das FORTRAN-Programm UPMPAR.FOR lautet:

```
C PROGRAMMNAME UPMPAR
      INTEGER UP,PAR
      BYTE FF,A,B
      DIMENSION PAR(3)
      FF=12
C DEFINITION DES PARAMETERBLOCKS
C ******************************
C *                            *
C *  LAENGE 80 X 18 ZEICHEN    *
      PAR(1)=X'F850'
      PAR(2)=X'F800'
      PAR(3)=X'5A0'
      WRITE (1,100)FF
100   FORMAT(' ',A1)
      A=19
      B=1
      CALL KURPOS(A,B)
      WRITE (1,120)
120   FORMAT(' ','========================================
     1 ==========================')
      A=21
      B=1
      CALL KURPOS(A,B)
      WRITE (1,130)
130   FORMAT(' ',FREIER TEIL ZUR BEDIENERFUEHRUNG')
      CALL UP(PAR(1))
C *** HL=adr PARAMETERBLOCK ***
      A=18
      B=1
      CALL KURPOS(A,B)
      WRITE (1,140)
140   FORMAT(' ',79X)
      A=19
      B=1
      CALL KURPOS(A,B)
      READ (1,150)K,L,M
```

```
150   FORMAT(3F10,3)
      GOTO 200
      END
      SUBROUTINE KURPOS(I,K)
      BYTE I,K,ESC,X,Y
      ESC=27
      X=I+127
      Y=K+127
      WRITE (1,110)ESC,X,Y
110   FORMAT(' ',3A1)
      RETURN
      END
```

Es ist zu gewaehrleisten, dass alle Parameter beim Aufruf in Anzahl, Typ und Laenge den benoetigten Parametern entsprechen.
FORTRAN-Funktionen geben in Abhaengigkeit vom Ergebnistyp zurueck:

LOGICAL in A,
INTEGER in HL,
REAL in *AC,
DOUBLE PRECISION in *DAC.

*AC und *DAC sind die Namen fuer die Adressen des niedrigsten Byte der Mantisse.

6.10. Ueberlagerung von Programmen

Es wird eine komplette Ueberlagerung von Hauptprogrammen im Nutzerspeicher realisiert (sequentiellle Verkettung). Das zuerst geladene Programm ruft ueber die FCHAIN-Anweisung ein zweites Programm, welches das erste ueberschreibt, etc.

Die Form der Anweisung lautet:

CALL FCHAIN (dateibezeichnung,Laufwerk)

Hierbei bedeuten:

dateibezeichnung
- Name des nachzuladenden Programms;
  Der Name muss 11 Stellen lang sein und in Hochkommas eingeschlossen werden.

Laufwerk
- Ist die Laufwerksnummer des Diskettenlaufwerks, von dem das Programm nachgeladen werden soll.
  Es muss eine ganze Zahl im Bereich von 0 bis 4 sein.
  1 gibt das Laufwerk A, 2 das Laufwerk B an usw. 0 ist fuer das aktuelle Laufwerk vorgesehen.

Die einzelnen Programme werden als Hauptprogramme erfasst, einzeln uebersetzt und gebunden. Zu beachten ist, dass die Ueberlagerung ab TPA-Beginn (100H) erfolgt.

4.2. Version 1.1 zur E/A-Bibliothek und Zusatzpaket 1/2"-Magnetbandunterstuetzung im C-Compiler

Anmerkung

Mit der Version 1.1 des Compilers wird eine neue E/A- Bibliothek bereitgestellt, die eine Anpassung an weitverbreitete C-Bibliotheken darstellt. Die bisherige Bibliothek ist noch als Teilmenge enthalten, so dass keine Aenderung vorhandener C-Quellprogramme notwendig ist.
Zusaetzlich wird eine Bibliothek bereitgestellt, die eine effektive Arbeit mit dem 1/2"- Magnetband auf dem Niveau der C- Sprache ermoeglicht.

Die nachfolgende Beschreibung wird bis zur naechsten Auflage der Dokumentation "Der C - Compiler fuer das Betriebssystem SCPX 1526" vorab in dieser Schrift veroeffentlicht. Sie beinhaltet die veraenderte Standard- E/A- Bibliothek, sowie die Zusatzkomponente fuer die Arbeit mit dem 1/2"- Magnetband. Zur besseren Einordnung wird eine zur C- Dokumentation passfaehige Gliederung verwendet.
Ausserdem wird als Anlage eine Zusammenfassung aller Bibliotheksfunktionen bereitgestellt.

Aenderungen zum Compiler

1. Die Kommandodatei **CM.SUB** wurde erweitert und zur Unterscheidung in **CC.SUB** umbenannt.
   Sie enthaelt jetzt den Aufruf des Assemblers und des Binders.
   Ein zusaetzlicher Parameter ermoeglicht die Ausgabe der Fehlerliste des Compilers auf den Drucker.
   Zur Uebersetzung eines C- Quellprogrammes in ein abarbeitbares Programm mit Drucken der Fehlerliste ist folgendes Kommando anzugeben:

   A>SUBM CC d:dateiname P

2. Die Standardmakrobibliothek **STD.H** wurde vereinfacht und in **STDIO.H** umbenannt.

3. Das Listprogramm **CLIST** wird unter Verwendung der neuen Standardmakrobibliothek als Demonstrationsprogramm bereitgestellt.

12. Die Standard-E/A-Bibliothek des Compilers

Da die C-Sprache selbst keine Anweisungen zur Ein-/Ausgabe, bzw. zur Dateiarbeit und Programmsteuerung enthaelt, sind diese Aufgaben ausschliesslich ueber Funktionen zu realisieren. Der Compiler CC 1520 bietet deshalb eine Bibliothek an, die alle betriebssystem- und maschinenabhaengigen Funktionen (E/A, Dateiarbeit, Speicherplatzverwaltung u.ae.) und eine Reihe universeller Funktionen zur Programmierunterstuetzung (Zeichenkettenverarbeitung, Typkonvertierung u.ae.) enthaelt.

Nachfolgend werden alle Bibliotheksfunktionen hinsichtlich ihrer Parameter, ihrer Semantik und ihrer Ergebnisse aufgelistet und teilweise durch einfache Beispiele erlaeutert. Da diese Funktionen zu jedem C-Compiler bereitgestellt werden, ist eine hohe Portabilitaet gewaehrleistet. Auf Unterschiede zum "Standard" wird in der Anlage C hingewiesen.
Fuer die Arbeit mit den E/A-Funktionen werden einige Systemkonstanten bzw. eine Datei-E/A-Struktur benoetigt, die in der Standardmakrobibliothek "stdio.h" definiert sind.
Nachfolgend wird diese Datei beschrieben und einige Begriffe erlaeutert.

12.1. Begriffsdefinitionen

(1) Dateiname

Der Dateiname (fname) ist eine Zeichenkette (d.h. ein durch NULL begrenztes Feld von Zeichen bzw. ein Zeiger auf diese Zeichenkette bei der Verwendung als Funktionsparameter). Er kann aus Ziffern und Buchstaben (Klein- oder Grossbuchstaben, nicht gemischt) bestehen. Das erste Zeichen ist immer ein Buchstabe, max. sind 8 Zeichen zugelassen, wahlweise gefolgt von einem Dezimalpunkt und max. 3 Buchstaben fuer den Dateityp. Das Voranstellen eines Laufwerksnamens oder die Definition eines beliebigen logischen Geraetes, gefolgt von einem Doppelpunkt, ist moeglich.
(--) Systemdokumentation zum Betriebssystem SCP)

(2) Dateibezeichner

Der Dateibezeichner (fd) ist eine kurze ganze Zahl (Typ short), deren Wert nur bei auftretenden Fehlern negativ wird. Waehrend der Laufzeit eines Programmes wird die Arbeit mit einer Datei ueber den beim Eroeffnen zugewiesenen Dateibezeichner realisiert.

(3) Dateimodus

Der Dateimodus (mode) ist ebenfalls eine kurze ganze Zahl (Typ short). Er kennzeichnet die auszufuehrende Operation mit einer Datei .

Es werden drei Modi spezifiziert:
- mode == 0 - fuer Lesen
- mode == 1 - fuer Schreiben
- mode == 2 - fuer Aendern

zusaetzlich
- mode ==-1 - fuer gepuffertes Schreiben (nur bei Arbeit mit der Datei-E/A-Struktur)

(4) Binaer- und Textdateien

Eine Binaerdatei (binary file) ist eine periodische Folge von Zeichen. Dabei existiert keine Satzstruktur und alle Kodes sind zugelassen.

Eine Textdatei (text file) ist prinzipiell vergleichbar mit einer Binaerdatei mit zwei Einschraenkungen:
Es sind nur Zeichen des ASCII-Kodes (druckbarer Text) einschliesslich Steuerzeichen (LF, FF, CR) zugelassen. Eine Satzstruktur wird durch die Einfuegung von NL erreicht, wobei ein solcher Satz nicht laenger als 512 Zeichen (einschl. NL) sein darf.

In einem C-Programm werden alle Ein- /Ausgabefunktionen ueber Dateien realisiert, das heisst, auch die E/A-Geraete werden intern wie Dateien behandelt.
Jedes C-Programm arbeitet mit 3 Textdateien, die normalerweise die Standard-E/A-Geraete (Tastatur und Bildschirm) repraesentieren und fuer die folgende Dateibezeichner definiert sind:

- **STDIN** (fd = 0) - Standardeingabedatei fuer Lesen (Tastatur)
- **STDOUT** (fd = 1) - Standardausgabedatei fuer Schreiben (Bildschirm)
- **STDERR** (fd = 2) - Standardfehlerausgabe fuer Schreiben (Bildschirm)

(5) Die Datei-E/A-Struktur

Eine komfortable Dateiarbeit wird ueber eine Datei-E/A-Struktur (FILE) realisiert, die in der Standardmakrobibliothek definiert ist und von den meisten E/A-Funktionen als Datei-Steuer-Block genutzt wird.

Strukturelemente sind:

- der Dateibezeichner (_fd)
  legt fest, fuer welche Datei eine Operation ausgefuehrt werden soll;
- ein Zeichenzaehler (_nleft)
  beinhaltet die Anzahl der Zeichen, die in den Puffer eingegeben werden bzw. sich im Puffer befinden;
  fuer _nleft=0 --) Initialisierung des Puffers;
  fuer _nleft=-1 --) das Dateiende ist erreicht und damit kein weiterer Zugriff moeglich

- der Dateimodus (_mode)
  legt fest, welche Operation ausgefuehrt werden soll
- ein Zeiger (_*pnext),
  verweist bei der Eingabe auf das naechste zu lesende Zeichen; bei der Ausgabe dient er zum Ketten der E/A-Strukturen, falls mehrere benoetigt werden;
- ein Textpuffer der Laenge 512 Byte (_buf[BUFSIZE])

Fuer die Standardtextdateien sind ebenfalls E/A-Strukturen vordefiniert, die in der Bibliothek als externe Variable vereinbart sind:

extern FILE stdin;
E/A-Puffer, der fuer die Eingabe von **STDIN** initialisiert ist

extern FILE stdout;
E/A-Puffer, der fuer die Ausgabe auf **STDOUT** initialisiert ist

extern FILE stderr;
E/A-Puffer, der fuer die Ausgabe auf **STDERR** initialisiert ist

12.2. Die Standardmakrobibliothek

Jedes C-Programm, welches bestimmte Systemkonstanten benutzt bzw. einige nuetzliche Makros verwendet, muss am Anfang die Anweisung:
#include "stdio.h"
zum Einfuegen der Standardmakrobibliothek enthalten.
Diese Datei besteht aus den Definitionen wichtiger Systemparameter sowie einfacher Makros, die wie Funktionen der E/A-Bibliothek verwendet werden koennen.

Systemparameter

```
#define STDIN  0          Standardeingabe (fd = 0)
#define STDOUT 1          Standardausgabe (fd = 1)
#define STDERR 2          Standardfehlerausgabe (fd = 2)
#define BUFSIZE 512       Laenge des Standard-E/A-Bereichs

#define BWRITE (-1)       gepuffertes Schreiben (mode == -1)
#define READ 0            Lesen (mode == 0)
#define WRITE 1           Schreiben (mode == 1)
#define UPDATE 2          Aendern (mode == 2)
#define EOF (-1)          Dateiendekennzeichen

#define YES 1             Wahrheitswert WAHR
#define NO  0             Wahrheitswert FALSCH
#define NULL 0            Kennzeichen
```

13. Funktionen zur Programmsteuerung

Nachfolgende Funktionen unterstuetzen das Abarbeiten eines C-Programmes.
Waehrend der Laufzeit eines Programmes wird immer die Funktion main() aufgerufen, die den Eintritt in ein Programm steuert.
Durch exit() kann ein Programm verlassen werden.

Ausserdem koennen mit getflags() die Optionen einer Kommandozeile verarbeitet werden.

13.1. main - Beginn der Programmabarbeitung

PARAMETER:

```
main (argc, argv)
   int argc;          /* Zaehler fuer Argumente */
   char **argv;       /* Verweis auf die Argumente */
```

SEMANTIK:

main ist eine Funktion, die immer zu Beginn der Abarbeitung eines C - Programmes aufgerufen wird.
Insbesondere werden die drei Standard-Textdateien **STDIN**, **STDOUT** und **STDERR** eroeffnet und die Kommandozeile ausgewertet.

Der Aufruf eines C- Programmes wird durch ein Kommando der Form

arg_0 [arg_1 arg_2 ... arg_n]

realisiert.
Dabei bezeichnet arg_0 den Namen des ausfuehrbaren Programmes. Weitere aktuelle Parameter arg_1...arg_n koennen z.B. Optionen zur Praezisierung bestimmter Aufgaben, Dateinamen, die das Programm verarbeiten soll, u.ae. sein.
Kommandoparameter sind also Zeichenketten, die durch Leerzeichen voneinander getrennt sind.

Um eine derartige Kommandozeile zu verarbeiten, sind an die main- Funktion zwei Argumente zu uebergeben:

argc stellt einen Zaehler fuer die Kommandoparameter dar, ist also mind. 1
argv ist die Adresse eines Zeigerfeldes, wobei jedes Element die Anfangsadresse eines Parameters enthaelt

Besonderheit:

Zusaetzlich (d.h., nicht in argc bzw. argv enthalten) koennen in einer Kommandozeile zwei Parameter angegeben werden, mit deren Hilfe die Standardtextdateien **STDIN** und **STDOUT** mit anderen peripheren Geraeten oder nutzerspezifischen Dateien verbunden werden koennen (File-umlenkung). Die Standardfehlerausgabe **STDERR** bleibt immer mit dem Bildschirm verbunden.

Beginnt eine Zeichenkette mit "<", so wird diese Kette als Name einer Datei (oder eines logischen Geraetes) interpretiert, die zum Lesen eroeffnet wird. (STDIN)
Beginnt eine Zeichenkette mit ">", so wird diese als Name einer Datei (oder eines logischen Geraetes) interpretiert, die zum Schreiben eroeffnet wird. (STDOUT)

Die Programmabarbeitung wird beendet, wenn das Ende der main()-Funktion erreicht ist oder explizit die Funktion exit() aufgerufen wird. In jedem Falle wird in einem Statusbyte angezeigt, ob die Programmabarbeitung erfolgreich war oder nicht.

ERGEBNIS:
main liefert den Wahrheitswert WAHR (=1) fuer die erfolgreiche Programmabarbeitung oder FALSCH (=0).

BEISPIEL:
Das Programm "TEST.C" soll ein Zeichen aus der Kommandozeile zum Standardausgabegeraet (Bildschirm) schreiben.

```
#include "stdio.h"   /* Einfuegen der Standardmakrobibl. */
main(argc,argv)
   int argc;
   char **argv;      /* identisch zu char *argv[]; */
   {
   if(argc < 2)
      return (NO);       /* keine Argumente vorhanden */
   else
      return (write (STDOUT,av[1],1) == 1);
   }
```

Durch die Kommandozeile:
TEST * <ENTER>
wird das Zeichen '*' auf dem Bildschirm ausgegeben.

Soll das Zeichen gedruckt werden, ist die Kommandozeile wie folgt zu modifizieren:

TEST * >LST:<ENTER>
(LST: ... Geraetebezeichnung fuer Drucker)

13.2. exit - Beenden der Programmabarbeitung

PARAMETER:
```
exit (success)
   int success;
```

SEMANTIK:
exit schliesst alle eroeffneten Dateien ab und beendet die Programmabarbeitung.
Als Argument **success** wird ein ganzzahliger Wert ungleich Null fuer die ordnungsgemaesse Programmabarbeitung, oder gleich Null fuer das Gegenteil, angegeben.

ERGEBNIS:

exit liefert kein Ergebnis, das heisst, die Funktion kehrt nie an die Stelle ihres Aufrufes zurueck.

BEISPIEL:

Verlassen eines Programmes nach Fehlerausgabe, wenn eine Datei "file" nicht eroeffnet werden kann.

```
...
if ((fd = open("file", READ, 0)) < 0)
    {
    write (STDERR, "can't open file\n", 16);
    exit (0);
    }
...
```

13.3. getflags - Verarbeiten der Optionen einer Kommandozeile

PARAMETER:

```
char *getflags (pargc,pargv,format,arg_1,arg_2,...arg_n)
   int *pargc;     /* Anzahl der Argumente */
   char ***pargv; /* Verweis auf die Argumente */
   char *format;  /* Formatkette */
```

SEMANTIK:

getflags entschluesselt die Optionen, die als Argumente in einem Kommando auftreten, und interpretiert sie ueber eine Formatkette format als Konstanten, Texte o.ae. Die Argumente, die sich auf die Parameter der main-Funktion beziehen, werden durch den Zaehler pargc und den Verweis auf die Argumente selbst pargv spezifiziert. Es wird vorausgesetzt, dass das erste Argument der Kommandoname ist und uebergangen wird.

Jedes folgende Argument wird als Option (Flag) interpretiert, wenn es

a) mit einem Vorzeichen "+" oder "-" beginnt und
b) nicht das Zeichen "-" oder "--" ist.

Die Formatkette format ist eine Folge von Bezeichnern, die festlegen, wie jedes Argument arg_1...arg_n interpretiert werden soll. Sie werden durch Komma oder Leerzeichen voneinander getrennt.
Ein solcher Bezeichner besteht aus dem Namen und evtl. einem Formatzeichen.

Formatzeichen haben folgende Bedeutung:

"*" - Als Kommandoparameter ist eine beliebige Zeichenfolge ist zugelassen. Es wird ein Zeiger auf das entsprechende Argument arg_n gebildet.

"?" - Es ist ein beliebiges Zeichen zugelassen, dessen Bytewert als Ganzzahlkonstante interpretiert wird.

"#" - Es wird der Zahlenwert des entsprechenden Arguments ermittelt, wobei fuer dieses nur Zeichen entsprechend der Darstellung von Ganzzahlkonstanten zugelassen sind. Das heisst, die Zeichenfolge wird als dezimale, oktale oder hexadezimale Konstante interpretiert.

ERGEBNIS:
getflags liefert einen Zeiger auf die uebrigbleibende Argumentkette, wenn ein Fehler aufgetreten ist.
Koennen alle Flags ordnungsgemaess verarbeitet werden, wird NULL zurueckgegeben.

BEISPIEL:
Verarbeiten der Optionen aus dem Listprogramm CLIST (--> Anlage E)

```
getflags (&argc, &argv, "h, n, i#, b#, e#", \
          &h_flag, &n_flag, &prefix, &begin, &ende)
```

die Kommandozeile koennte wie folgt aussehen:
A>clist -b100 -e200 (dateibezeichnung)
(Es werden die Zeilen 100 bis 200 der benannten Datei ausgegeben)

14. Standard-E/A-Funktionen

Die einfachste Moeglichkeit des Datenaustausches erfolgt ueber die Standardtextdateien **STDIN**, **STDOUT** und **STDERR** bzw. die zugeordneten E/A-Strukturen.
Durch eine Umlenkung dieser Dateien zu anderen logischen Geraeten oder nutzerspezifischen Dateien sind die Standard-E/A-Funktionen vielseitig anwendbar.

Es stehen sowohl Funktionen zur formatgesteuerten Texte in- und -ausgabe scanf(), printf() und errfmt() zur Verfuegung.
Zum Verarbeiten einzelner Textzeichen und Textzeilen werden getchar() und putchar() bzw. getlin() und putlin() bereitgestellt.

14.1. scanf - Formatgesteuerte Eingabe

PARAMETER:

```
int scanf (format, arg_1, arg_2, ...arg_n)
   char *format;       /* Formatkette */
```

SEMANTIK:
scanf liest eine formatierte Eingabezeile vom Standardeingabepuffer **stdin**, interpretiert sie entsprechend dem Format aus der Zeichenkette **format** und legt die Ergebnisse in den entsprechenden Argumenten **arg_1,arg_2,...arg_n** ab. Diese Argumente muessen Zeiger (oder Speicheradressen) sein.

Die Formatkette darf folgendes enthalten:
- beliebige Textzeichen ausser '%' einschl. Trenn- und Steuerzeichen;
  diese Zeichen werden ignoriert und dienen nur der Kommentierung
- Formatspezifikationen

Eine Formatspezifikation legt fest, wie ein bestimmter Eingabebereich interpretiert und konvertiert werden soll. Sie haben folgende Form:

%[*](formatzeichen)
Mit dem Zeichen '%' wird ein Formatelement eingeleitet. Durch eine Ziffernfolge * kann wahlweise die maximale Laenge des Eingabebereiches begrenzt werden.
Wie die Eingabe interpretiert werden soll, legt das (formatzeichen) fest. Normalerweise wird das Ergebnis in der Variablen abgespeichert, auf die das zugehoerige Argument verweist. Durch Unterdruecken der Zuweisung (Formatzeichen "x") wird die Eingabe zwar gelesen, es erfolgt jedoch keine Zuordnung.

Folgende Formatzeichen sind zugelassen :

c ... Bytewert
i ... ganze Zahl
s ... kurze ganze Zahl
l ... lange ganze Zahl
a ... ASCII
h ... Hexadezimal
o ... Oktal
u ... ohne Vorzeichen
p ... Zeichenkette, die mit '\0' abgeschlossen wird
d ... Gleitkommazahl doppelte Genauigkeit
f ... Gleitkommazahl einfache Genauigkeit
x ... Eingabe ueberlesen

Diese Formatzeichen koennen auch kombiniert auftreten (z.B. %ac --) ein ASCII- Zeichen wird erwartet)

Die Eingabe wird als Zeichenkette definiert. Die Elemente werden durch NULL, durch Steuerzeichen (Leerzeichen, Tabulator) oder eine angegebene Feldbreite begrenzt.
Die Funktion wird beendet, wenn die Formatkette abgearbeitet wurde oder ein Formatfehler vorliegt.

ERGEBNIS:
scanf liefert die Anzahl der richtig konvertierten Argumente. Bei Erreichen des Dateiendes wird EOF zurueckgegeben.

BEISPIEL:
Folgende Eingabezeile soll unterschiedlich interpretiert werden:

123 0xFF 3.14 PI

```
/* Vereinbarungen */
int i, n;
double pi;
char text[5];

scanf("%i %h %d %p", &i, &n, &pi, text);

/* --> i=123; n=255=0xff; pi=3.14; text[]="PI"; */

scanf("%2i %x %p %x", &n, text);

/* --> n=12; text[]="3.14"; */

...
```

14.2. printf - Formatgesteuerte Ausgabe

PARAMETER:

```
printf (format, arg_1, arg_2, ...arg_n)
   char *format;       /* Formatkette */
```

SEMANTIK:

printf konvertiert und formatiert ihre Argumente unter Steuerung der Zeichenkette format und schreibt sie in den Standardausgabepuffer stdout.
Sie stellt im Prinzip die Umkehrfunktion zu scanf() dar.
Die Formatkette kann aus einfachen Textzeichen, die einfach in die Ausgabe kopiert werden, und Formatspezifikationen bestehen.

Formatspezifikationen haben folgende Form:

```
%[+|-[c[#]]](formatzeichen)
```

Durch eine Ziffernfolge # kann wiederum eine bestimmte Ausgabebreite erzwungen werden. Sie kann aus zwei durch Dezimalpunkt getrennte Ziffernfolgen bestehen. Die erste Zahl definiert die maximale Ausgabebreite, die zweite Zahl gibt entweder die minimale Ausgabebreite oder die Stellen nach dem Komma (bei Gleitkommadarstellung) an.
Die Zeichen + bzw.- bewirken ein Auffuellen links bzw. rechts mit einem beliebigen Textzeichen c (wenn c fehlt, wird mit Leerzeichen aufgefuellt).

Folgende Formatzeichen sind erlaubt:

```
c ... Bytewert
i ... ganze Zahl (2Byte)
s ... kurze ganze Zahl
l ... lange ganze Zahl
a ... ASCII
h ... Hexadezimal
o ... Oktal
u ... ohne Vorzeichen
p ... Zeichenkette
```

```
d ... Gleitkommazahl (Exponentialform)
f ... Gleitkommazahl (mit Dezimalpunkt)
x ... unterdruecken der Ausgabe

Das Zeichen '%' selbst wird durch %% ausgegeben.
```

ERGEBNIS:

```
printf liefert keinen Wert.
```

BEISPIEL:

Die Anzahl der Fehler in einer Datei soll ausgegeben werden.

```
int n = 10;                 /* Zaehler fuer Fehler */
char *fname = "TEST.DOC;    /* Dateiname */
...
printf("%i Fehler in der Datei %p\n", n, fname);
```

Auf dem Bildschirm wuerde in diesem Fall erscheinen:
10 Fehler in der Datei TEST.DOC

14.3. errfmt - Formatgesteuerte Fehlerausgabe

PARAMETER:

```
errfmt (format, arg_1, arg_2, ...arg_n)
   char *format;        /* Formatkette */
```

SEMANTIK:

errfmt schreibt eine formatierte Ausgabezeile vom Standardausgabepuffer stderr in der gleichen Art und Weise wie die Funktion printf().

ERGEBNIS:

```
errfmt liefert keinen Wert.
```

14.4. getchar - Lesen eines Textzeichens

PARAMETER:

```
int getchar()
```

SEMANTIK:

getchar liest das naechste Eingabezeichen von STDIN.

ERGEBNIS:

getchar liefert das gelesene Zeichen.
Das Dateiende wird durch das Kennzeichen EOF (=-1) angezeigt.

14.5. putchar - Schreiben eines Textzeichens

PARAMETER:

```
int putchar(c)
   int c;          /* auszugebendes Zeichen */
```

SEMANTIK:
putchar gibt das Zeichen c nach **STDOUT** aus.

ERGEBNIS:
putchar liefert das Zeichen c.

BEISPIEL:
Zeichenweises Kopieren einer Datei.

```
...
while (putchar(getchar()) != EOF)  ;
```

14.6. getlin - Lesen einer Textzeile

PARAMETER:

```
int getlin (s, n)
   char *s;          /* Textpuffer */
   int n;            /* Zeichenanzahl */
```

SEMANTIK:
getlin liest eine Textzeile **STDIN** zu einem Puffer s der Laenge n.
Es werden solange Zeichen gelesen, bis
a) eine Zeilenschaltung ('\n') erkannt wird,
b) das Dateiende (EOF) erreicht ist oder
c) n Zeichen gelesen sind.

ERGEBNIS:
getlin liefert die Anzahl der gelesenen Zeichen, also einen Wert zwischen 1 und n, solange bis das Dateiende erreicht wurde. Von dieser Stelle an wird immer der Wert Ø zurueckgegeben.

14.7. putlin - Ausgeben einer Textzeile

PARAMETER:

```
int putlin (s, n)
   char *s;          /* Textpuffer */
   int n;            /* Zeichenanzahl */
```

SEMANTIK:
putlin schreibt eine Textzeile der Laenge n von einem Puffer s nach **STDOUT** aus.

ERGEBNIS:
putlin liefert die Anzahl der tatsaechlich geschriebenen Zeichen.

BEISPIEL:
Zeilenweises Kopieren einer Datei.

```
char buf[BUFSIZE];
...
while (putlin(buf, getlin(buf, BUFSIZE)))
      ;
```

15. Dateiarbeit

Eine komfortable Moeglichkeit zum Verarbeiten von nutzereigenen Dateien ist die Verwendung der E/A-Struktur **FILE**, die in der Standardmakrobibliothek deklariert ist.
Es ist moeglich, die Dateien ueber ihren Namen zu eroeffnen und zu schliessen. Dazu werden die Funktionen **fopen()**, **fcreate()** und **fclose()** angeboten. Diese Funktionen liefern die Adresse einer E/A-Struktur, einen sogenannten Dateizeiger (Filepointer fp).
Einmal erstellt, kann die E/A-Struktur durch formatierte Text-E/A **fscanf()** und **fprintf()** genutzt werden. Einzelne Zeichen oder Zeilen werden durch die Funktionen **getc()** und **putc()** bzw. **getl()** und **put()** verarbeitet.
Diese Funktionen sind prinzipiell mit den Standard-E/A-Funktionen vergleichbar mit dem Unterschied, dass fuer die benutzten Dateien jeweils eine E/A-Struktur vereinbart werden muss.

15.1. finit - Initialisieren einer E/A - Struktur

PARAMETER:

```
FILE *finit (fp, fd, mode)
   FILE *fp;          /* E/A-Struktur */
   short fd;          /* Dateibezeichner */
   short mode;        /* Modus */
```

SEMANTIK:

finit initialisiert eine E/A-Struktur **fp** fuer die Arbeit mit der durch **fd** spezifizierten Datei.

ERGEBNIS:

finit liefert die Anfangsadresse des E/A - Puffers oder den Wert 0, falls ein Fehler auftritt.

15.2. fcreate - Erzeugen einer Datei und Initialisieren der E/A - Struktur

PARAMETER:

```
FILE *fcreate (fp, fname, mode)
   FILE *fp;          /* E/A-Struktur */
   char *fname;       /* Dateiname */
   short mode;        /* Modus */
```

SEMANTIK:

fcreate eroeffnet eine neue Datei mit dem spezifizierten Namen **fname** und initialisiert die E/A-Struktur fuer die weitere Arbeit mit der Datei.
Der Modus **mode** muss dabei einen der Werte fuer READ (=0), WRITE (=1) oder BWRITE (=-1) haben.

ERGEBNIS:

fcreate liefert einen Zeiger **fp** auf die E/A-Struktur oder den Wert 0, falls ein Fehler auftritt.

BEISPIEL:
Erzeugen einer Datei "file" und Eroffnen der E/A - Struktur "infile". Ausgabe einer Fehlermeldung, wenn die Datei nicht ordnungsgemaess eroeffnet werden kann.

```
FILE infile;
...
if (fcreate(&infile, "file", READ))
    errfmt("can't create file\n");
...
```

15.3. fopen - Eroeffnen einer Datei und Initialisieren der E/A - Struktur

PARAMETER:

```
FILE *fopen (fp, fname, mode)
   FILE *fp;        /* E/A-Struktur */
   char *fname;     /* Dateiname */
   short mode;      /* Modus */
```

SEMANTIK:
fopen eroeffnet eine Datei mit dem spezifizierten Namen fname und initialisiert die E/A-Struktur fuer die weitere Arbeit mit der Datei.
Der Modus mode muss dabei einen der Werte fuer READ (=0), WRITE (=1) oder BWRITE (=-1) haben.

ERGEBNIS:
fopen liefert einen Zeiger fp auf die E/A-Struktur oder den Wert 0, falls ein Fehler auftritt.

15.4. fclose - Abschliessen einer Datei

PARAMETER:

```
FILE *fclose (fp)
   FILE *fp;        /* E/A-Struktur */
```

SEMANTIK:
fclose schliesst die durch die E/A-Struktur fp initialisierte Datei ab.

ERGEBNIS:
fclose liefert den Zeiger auf die freigewordene Struktur oder einen negativen Wert.

BEISPIEL:
Eroeffnen einer Datei "file" und Abschliessen nach der Bearbeitung.

```
FILE outfile; char file[];
...
fopen(&outfile, file, WRITE);
... lesen und verarbeiten ...
fclose(&outfile);
```

15.5. fscanf - Formatgesteuerte Eingabe

PARAMETER:

```
int fscanf (fp, format, arg_1, arg_2, ...arg_n)
   FILE *fp;        /* E/A-Struktur */
   char *format;    /* Formatkette */
```

SEMANTIK:

fscanf liest eine formatierte Eingabezeile von der durch die E/A-Struktur fp bezeichnete Datei, indem die Formatkette format mit den entsprechenden Argumenten arg_1, arg_2,...arg_n verbunden wird.
Die Verarbeitung erfolgt wie in scanf().

ERGEBNIS:

fscanf liefert die Anzahl der richtig zugewiesenen Argumente oder EOF (-1), wenn das Ende der Datei erreicht ist, noch bevor etwas kopiert wird.

14.6. fprintf - Formatgesteuerte Ausgabe

PARAMETER:

```
fprintf (fp, format, arg_1, arg_2, ...arg_n)
   FILE *fp;        /* E/A-Struktur */
   char *format;    /* Formatkette */
```

SEMANTIK:

fprintf schreibt eine formatierte Ausgabezeile auf die durch die E/A-Struktur fp bezeichnete Datei, indem die Formatkette format mit den entsprechenden Argumenten arg_1, arg_2,...arg_n verbunden wird.
Die Verarbeitung erfolgt wie in printf().

ERGEBNIS:

fprintf liefert keinen Wert.

15.7. getc - Lesen eines Zeichens

PARAMETER:

```
int getc (fp)
   FILE *fp;        /* E/A-Struktur */
```

SEMANTIK:

getc liest das naechste Eingabezeichen von der durch die E/A - Struktur fp gesteuerten Datei.

ERGEBNIS:

getc liefert das gelesene Zeichen.
Das Dateiende wird durch das Kennzeichen EOF (-1) angezeigt.

15.8. putc - Ausgeben eines Zeichens

PARAMETER:

```
int putc (fp, c)
   FILE *fp;        /* E/A-Struktur */
   int c;           /* Textzeichen */
```

SEMANTIK:

putc gibt das Zeichen c auf die durch die E/A-Struktur fp bezeichnete Datei aus.
Es wird vorausgesetzt, dass die E/A-Struktur fuer Schreiben oder Aendern eroeffnet wurde.

ERGEBNIS:

putc liefert das Zeichen c.

BEISPIEL:

Zeichenweises Kopieren einer Datei.

```
FILE infile, outfile;
...
while (putc(&outfile, getc(&infile)) != EOF)
      ;
...
```

15.9. getl - Lesen einer Textzeile

PARAMETER:

```
int getl (fp, s, n)
   FILE *fp;        /* E/A-Struktur */
   char *s;         /* Eingabepuffer */
   int n;           /* Pufferlaenge */
```

SEMANTIK:

getl liest eine Textzeile von der durch die E/A-Struktur fp bezeichneten Datei zu einem Puffer s der Laenge n.
Es werden solange Zeichen gelesen, bis
   a) eine Zeilenschaltung ('\n') erkannt wird,
   b) das Dateiende (EOF) erreicht ist oder
   c) n Zeichen gelesen wurden.

ERGEBNIS:

getl liefert die Anzahl der kopierten Zeichen, also einen Wert zwischen 1 und n, solange bis das Dateiende erreicht würde. Von dieser Stelle an wird immer der Wert Ø zurueckgegeben.

15.10. putl - Ausgeben einer Textzeile

PARAMETER:

```
int putl (fp, s, n)
   FILE *fp;        /* E/A-Puffer */
   char *s;         /* Textpuffer */
   int n;           /* Pufferlaenge */
```

SEMANTIK:
putl kopiert eine Textzeile der Laenge **n** von einem Puffer **s** zu der durch die E/A-Struktur **fp** bezeichneten Datei.

ERGEBNIS:
putl liefert die Anzahl der kopierten Zeichen.

BEISPIEL:
Zeilenweises Kopieren einer Datei.

```
FILE infile, outfile;
char *buf;
...
while(putl (&outfile, buf, getl(&infile, buf, BUFSIZE)))
      ;
...
```

16. Systemschnittstelle

Die Schnittstelle zum Betriebssystem SCPX wird ueber einfache E/A-Funktionen realisiert.
Bevor eine Datei gelesen bzw. in eine Datei geschrieben werden kann, muss diese erst eroeffnet werden. Dazu verwendet man die Funktionen **open()** und **create()**. Vom Betriebssystem wird dann ein Dateibezeichner **fd** bereitgestellt, der die Arbeit mit der Datei gewaehrleistet. **close()** gibt diesen Dateibezeichner nach dem Schliessen der Datei zurueck.
Dateibezeichner werden ebenfalls von den Funktionen **read()** und **write()** zum Bytetransport zwischen Datei und Speicher benutzt.
Die Datei-E/A ist normalerweise sequentiell. Jedes Lesen oder Schreiben erfolgt direkt nach der vorhergehenden Operation. Ist es jedoch erforderlich, kann durch **lseek()** in einer Datei direkt positioniert werden.
Diese E/A-Operationen koennen natuerlich auch auf die reservierten Standarddateien mit den Bezeichnern **STDIN**, **STDOUT** und **STDERR** angewendet werden.
Ausserdem koennen Dateien durch **unlink()** aus dem Diskettenverzeichnis geloescht werden.
Die direkte Arbeit mit dem Betriebssystem SCPX, das heisst, mit den Funktionen des BDOS, wird durch die Funktion **scp()** ermoeglicht.

16.1. create - Erzeugen einer Datei

PARAMETER:

```
short create (name, mode, rsize)
   char *name;      /* Dateiname */
   short mode;      /* Modus */
   int rsize;       /* Kennzeichen fuer Dateiart */
```

SEMANTIK:

create erzeugt eine neue Datei mit dem spezifizierten Namen **name**. Existiert eine solche Datei bereits, wird diese auf Laenge Ø gebracht.
Die Datei wird in Abhaengigkeit vom Zugriffsmodus **mode** eroeffnet.
Das Kennzeichen **rsize** legt fest, ob es sich um eine Binaerdatei (rsize = 0) oder eine Textdatei (rsize = 1) handeln soll.
Bei einer Binaerdatei werden alle "CR" und "NUL" bei der Eingabe geloescht und ein "CTRL-Z" wird an das Dateiende angefuegt.

ERGEBNIS:

create liefert den Dateibezeichner **fd** fuer die eroeffnete Datei oder einen negativen Wert, falls ein Fehler auftritt.

16.2. open - Eroeffnen einer Datei

PARAMETER:

```
short open (name, mode)
   char *name;      /* Dateiname */
   short mode;      /* Modus */
```

SEMANTIK:

open eroeffnet eine Datei mit dem spezifizierten Namen **name** und weist dieser einen Dateibezeichner **fd** zu.
In Abhaengigkeit des uebergebenen Modus **mode** wird die Datei zum Lesen (mode == Ø), Schreiben (mode == 1) oder Aendern (mode == 2) eroeffnet.

ERGEBNIS:

open liefert einen Dateibezeichner **fd** fuer die eroeffnete Datei oder einen negativen Wert, falls ein Fehler auftritt.

16.3. close - Abschliessen einer Datei

PARAMETER:

```
short close (fd)
   short fd;        /* Dateibezeichner */
```

SEMANTIK:

close schliesst die durch den Dateibezeichner **fd** bezeichnete Datei ab.

ERGEBNIS:

close gibt den freigewordenen Dateibezeichner **fd** oder einen ngativen Wert zurueck.

BEISPIEL:

Eroeffnen einer Datei "file" fuer einen Lesezugriff und Abschliessen nach Bearbeitung.

```
short infd;
...
if ((infd = open ("file",READ,1)) < 0)
     write (STDERR, "Fehler\n", 7);
... lesen und verarbeiten ...
close(infd);
```

16.4. read - Lesen einer Datei

PARAMETER:

```
int read (fd, buf, size)
   short fd;          /* Dateibezeichner */
   char *buf;         /* Textpuffer */
   int n;             /* Pufferlaenge */
```

SEMANTIK:

read liest die vorgegebene Anzahl Zeichen n von einer durch fd spezifizierten Datei in einen Puffer buf.

ERGEBNIS:

read liefert entweder den Wert 0, falls das Dateiende erreicht ist oder die Anzahl der gelesenen Zeichen. Ein negativer Wert kann nur bei fehlerhaftem Lesezugriff auftreten.

16.5. write - Schreiben in eine Datei

PARAMETER:

```
int write (fd, buf, size)
   short fd           /* Dateibezeichner */
   char *buf;         /* Textpuffer */
   int n;             /* Pufferlaenge */
```

SEMANTIK:

write schreibt die vorgegebene Anzahl Zeichen n von einem Puffer buf in die durch fd spezifizierte Datei.

ERGEBNIS:

write liefert die Anzahl der geschriebenen Zeichen, also einen Wert zwischen 0 und n, oder einen negativen Wert bei fehlerhafter Schreiboperation.

BEISPIEL:

Kopieren einer Datei vom Standardeingabe- zum Standardausgabegeraet und Abbruch bei Schreibfehler.

```
char *buf; int n;
...
while ((n = read(STDIN, buf, BUFSIZE)) > 0)
    if (write(STDOUT, buf, n) < 0)
        {
        write(STDERR,"write error\n",12);
        return(NO);
        }
```

16.6. lseek - Vorbereitung fuer den Direktzugriff

PARAMETER:

```
short lseek (fd, offset, origin)
   short fd;          /* Dateibezeichner */
   long offset;       /* Position */
   int origin;        /* Kennzeichen */
```

SEMANTIK:

lseek ermoeglicht das Positionieren in einer durch **fd** spezifizierten Datei, ohne diese Datei zu veraendern. Die Funktion setzt die aktuelle Position der Datei auf die Position **offset** , welche relativ zu der durch **origin** festgelegten Position ist:

- fuer origin = 0 vom Dateibeginn
- fuer origin = 1 von der aktuellen Position
- fuer origin = 2 vom Dateiende

ERGEBNIS:

lseek liefert den Dateibezeichner oder einen negativen Wert bei Fehler.

BEISPIEL:

In einer Datei soll eine Anzahl Zeichen von einer bestimmten Position gelesen werden.

```
...
get(fd, pos, buf, n)
    short fd;    /*Dateibezeichner*/
    int n;       /*Zeichenanzahl*/
    long pos;    /*Position*/
    char *buf;   /*Textpuffer*/
    {
    lseek (fd, pos, 01);
    return (read(fd, buf, n))
    }
...
```

16.7. unlink - Loeschen einer Datei

PARAMETER:

```
int unlink (fname)
   char *fname;     /* Dateiname */
```

SEMANTIK:

unlink loescht die Datei **fname** aus dem Diskettenverzeichnis.

ERGEBNIS:

unlink liefert den Wert Ø oder einen negativen Wert bei Fehler.

BEISPIEL:

Die Datei "TEST.DOC" soll aus dem Verzeichnis geloescht werden.

```
unlink ("TEST.DOC");
```

16.8. uname - Erzeugen eines universellen Dateinamens

PARAMETER:

```
char *uname()
```

SEMANTIK:

uname erzeugt eine Datei mit einem universellen Dateinamen der normalerweise mit keinem der ueblichen Dateinamen uebereinstimmt.
Der so erzeugte Name muss als erstes Argument zu create() oder open() angegeben werden.

ERGEBNIS:

uname liefert einen Zeiger auf den erzeugten Namen.

16.9. scp - Zugriff zum Betriebssystem

PARAMETER:

```
int scp (bc, de)
      int bc;
      char *de;
```

SEMANTIK:

scp bietet die Moeglichkeit, direkt die BDOS-Funktionen des SCPX aufzurufen.
Die Parameter bc und de werden direkt in die Prozessorregister BC und DE geladen und anschliessend wird ein Sprung an BDOS ausgefuehrt.
Wirkungsweise und Moeglichkeiten des BDOS sind der Systemunterlage fuer das SCPX "Anleitung fuer den Programmierer" zu entnehmen.

ERGEBNIS:

scp liefert den Wert der gerufenen BDOS-Funktion, das heisst, den Inhalt des Registers A in einer Int- Variablen.

16.10. in - Eingabe von einem Maschinenport

PARAMETER:

```
int in(port)
      char port;
```

SEMANTIK:

in ermoeglicht die Eingabe eines Bytewertes von einem Port port.

ERGEBNIS:

in liefert das gelesene Byte als Int- Wert, dessen hoeherwertiges Byte Null ist.

16.11. out - Ausgabe auf ein Maschinenport

PARAMETER:

```
out(port, out)
    char port, out;
```

SEMANTIK:

out ermoeglicht die Ausgabe eines Bytewertes out auf ein Port port.

ERGEBNIS:

out liefert keinen definierten Wert.

17. Speicherverwaltung

Die Bibliothek des Compilers unterstuetzt eine effektive Speicherplatzverwaltung.
Die Funktion alloc() fordert vom Betriebssystem nach Bedarf Speicherplatz an, wobei dieser nicht immer zusammenhaengend sein muss. Durch free() kann nicht mehr benoetigter Speicherplatz wieder zurueckgegeben werden.
Die Funktion buybuf() ist eine Erweiterung von alloc(), indem in den entsprechend zugewiesenen Speicherplatz sofort ein Puffer kopiert wird.

17.1. alloc - Zuweisen von Speicherplatz

PARAMETER:

```
char *alloc (nbytes, link)
   int nbytes;      /* Speicherlaenge */
   char *link;      /* Zeiger auf 1. Element */
```

SEMANTIK:

alloc weist Speicherplatz der Laenge nbytes zu, beginnend an der Stelle, die durch den Zeiger link bezeichnet wird. Der Speicherplatz wird nach Erfordernis ausgedehnt. Wenn er erschoepft ist, wird die Fehlermeldung "out of heap space" auf STDERR ausgegeben und das Programm verlassen.

ERGEBNIS:

alloc liefert einen Zeiger auf den Anfang des zugewiesenen Speicherplatzes.

17.2. buybuf - Zuweisen von Speicherplatz und Kopieren eines Puffers

PARAMETER:

```
char *buybuf (s, n)
   char *s;         /* Textpuffer */
   int n;           /* Pufferlaenge */
```

SEMANTIK:
buybuf weist Speicherplatz der Laenge n zu und kopiert anschliessend die n Zeichen aus dem Puffer s in diesen Speicherplatz.
Ist der Speicherplatz erschoepft, wird die Funktion verlassen.

ERGEBNIS:
buybuf liefert einen Zeiger auf den Anfang des zugewiesenen Speicherplatzes.

17.3. free - Freigabe von Speicherplatz

PARAMETER:
```
char *free (link)
   char *link;      /* Zeiger auf 1.Element */
```

SEMANTIK:
free gibt den durch link bezeichneten Speicherplatz wieder zurueck, der vorher durch einen Aufruf von **alloc()** angefordert wurde.

ERGEBNIS:
free liefert den Zeiger auf das erste Element des freigegebenen Speicherplatzes.

18. Verarbeiten von Zeichen und Zeichenketten

Um einige Programmieraufgaben effektiv und ohne grossen Aufwand zu loesen, enthaelt die Makro-bzw. E/A-Bibliothek eine Vielzahl Makros zur Zeichenanalyse sowie Funktionen zur Arbeit mit Puffern und Zeichenketten.

18.1. iswhite - Test auf nichtdruckbare Zeichen

PARAMETER:
```
iswhite (c)
```
(die Variable c kann jeden numerischen Wert annehmen)

SEMANTIK:
iswhite testet das als Parameter uebergebene Zeichen c auf seinen ASCII - Kode. Alle Werte unter 0X20 (' ') und ueber 0X7F ('del') sind nichtdruckbare Zeichen, d.h., auch die Zeichen NUL ('\0') und LF ('\n') werden in diese Kategorie eingeordnet.

ERGEBNIS:
iswhite liefert den Wahrheitswert WAHR (=1) oder FALSCH (=0).

BEISPIEL:
In einem Text sollen alle Steuerzeichen gezaehlt werden.

```
char c;      /*Textzeichen*/
int ws;    /*Zaehler fuer Steuerzeichen*/
...
if (c = iswhite(c))
   ++ws;
...
```

18.2. isdigit - Test auf Ziffer

PARAMETER:
isdigit (c)
(die Variable c kann jeden numerischen Wert annehmen)

SEMANTIK:
isdigit testet das als Parameter uebergebene Zeichen c auf seinen ASCII - Kode. Alle Werte von 0X30 bis 0X39 ('0' - '9') werden als Dezimalziffern gekennzeichnet.

ERGEBNIS:
isdigit liefert den Wahrheitswert WAHR (=1) oder FALSCH (=0).

18.3. isalpha - Test auf Buchstabe

PARAMETER:
isalpha (c)
(die Variable c kann jeden numerischen Wert annehmen)

SEMANTIK:
isalpha testet das als Parameter uebergebene Zeichen c auf seinen ASCII - Kode. Alle Werte von 0X41 bis 0X5A ('A'-'Z') bzw. 0X61 bis 0X7A ('a'-'z') werden als Buchstaben gekennzeichnet, unabhaengig von ihrer Gross- oder Kleinschreibung.

ERGEBNIS:
isalpha liefert den Wahrheitswert WAHR (=1) oder FALSCH (=0).

18.4. isupper - Test auf Grossbuchstabe

PARAMETER:
isupper (c)
(die Variable c kann jeden numerischen Wert annehmen)

SEMANTIK:
isupper testet das als Parameter uebergebene Zeichen c auf seinen ASCII - Kode. Alle Werte von 0X41 bis 0X5A ('A'-'Z') werden als Grossbuchstaben gekennzeichnet.

ERGEBNIS:
isupper liefert den Wahrheitswert WAHR (=1) oder FALSCH (=0).

18.5. islower - Test auf Kleinbuchstabe

PARAMETER:

```
islower (c)
( die Variable c kann jeden beliebigen Wert annehmen )
```

SEMANTIK:

islower testet das als Parameter uebergebene Zeichen c auf seinen ASCII - Kode. Alle Werte von 0X61 bis 0X7A ('a'-'z') werden als Kleinbuchstaben gekennzeichnet.

ERGEBNIS:

islower liefert den Wahrheitswert WAHR (=1) oder FALSCH (=0).

18.6. toupper - Wandlung in Grossbuchstaben

PARAMETER:

```
toupper (c)
(die Variable c kann jeden numerischen Wert annehemen)
```

SEMANTIK:

toupper testet das als Parameter uebergebene Zeichen c. Wird ein Kleinbuchstabe erkannt, wird er in einen Grossbuchstaben gewandelt. Alle anderen Zeichen bleiben unveraendert.

ERGEBNIS:

toupper liefert den konvertierten Grossbuchstaben oder das unveraenderte Zeichen.

18.7. tolower - Wandlung in Kleinbuchstaben

PARAMETER:

```
tolower (c)
(die Variable c kann jeden numerischen Wert annehemen)
```

SEMANTIK:

tolower testet das als Parameter uebergebene Zeichen c. Wird ein Grossbuchstabe erkannt, wird er in einen Kleinbuchstaben gewandelt. Alle anderen Zeichen bleiben unveraendert.

ERGEBNIS:

tolower liefert den konvertierten Kleinbuchstaben oder das unveraenderte Zeichen.

18.8. strlen - Ermitteln der Laenge einer Zeichenkette

PARAMETER:

```
int strlen (s)
   char *s;          /* Zeichenkette */
```

SEMANTIK:
strlen liest die Zeichenkette **s** und berechnet die Anzahl der Zeichen einschliesslich Endekennzeichen NUL (\Ø').

ERGEBNIS:
strlen liefert die Anzahl der Zeichen in der Kette.

BEISPIEL:
Ausgabe einer Zeichenkette unbekannter Laenge zum Standardausgabegeraet.

```
char *s;
...
write (STDOUT, s, strlen (s));
...
```

18.9. fill - Initialisieren eines Textpuffers

PARAMETER:
```
int fill (s, n, c)
   char *s,          /* Puffer */
        c;           /* Textzeichen */
   int n;            /* Anzahl der Zeichen */
```

SEMANTIK:
fill fuellt den Puffer **s** der Laenge **n** mit dem vorgegebenen Textzeichen **c**.

ERGEBNIS:
fill liefert die Anzahl der Zeichen.

BEISPIEL:
Loeschen eines 512 - Byte - Puffers auf NUL ('\0') und Schreiben in eine Datei

```
short fd; char *buf;
...
write (fd, buf, fill (buf,BUFSIZE,'\Ø'));
```

18.10. squeeze - Verdichten eines Textpuffers

PARAMETER:
```
int squeeze (s, n, c)
   char *s,          /* Puffer */
        c;           /* Textzeichen */
   int n;            /* Anzahl der Zeichen */
```

SEMANTIK:
squeeze loescht das Textzeichen **c** aus einem Puffer **s** der Laenge **n** und verdichtet gleichzeitig den Puffer.

ERGEBNIS:
squeeze liefert die verbleibende Anzahl der Zeichen, d.h. einen Wert zwischen Ø und **n**.

BEISPIEL:

Entfernen aller Leerzeichen aus einem 512-Byte-Puffer und Schreiben zum Standardausgabegeraet

```
char *buf;
...
write (STDOUT, buf, squeeze (buf, BUFSIZE, ' '));
```

18.11. strncpy - Kopieren Puffer

PARAMETER:

```
int strncpy (s1, s2, n)
   char *s1, *s2   /* Puffer */
   int n;          /* Anzahl der Zeichen */
```

SEMANTIK:

strncpy kopiert die n Zeichen des Puffers s2 in den Puffer s1. Beide Puffer koennen eine unterschiedliche Laenge aufweisen.

ERGEBNIS:

strncpy liefert die Anzahl der kopierten Zeichen.

18.12. strcpy - Kopieren Zeichenkette

PARAMETER:

```
int strcpy (s1, s2)
   char *s1, *s2;   /* Zeichenketten */
```

SEMANTIK:

strcpy kopiert die durch NUL abgeschlossene Zeichenkette s2 in die Zeichenkette s1.

ERGEBNIS:

strcpy liefert die Anzahl der kopierten Zeichen.

18.13. strcat - Kopieren mehrerer Zeichenketten hintereinander

PARAMETER:

```
char *strcat (ds, arg_1, arg_2, ...arg_n, NUL)
   char *ds,       /* Zielzeichenkette */
        *arg_1, *arg_2, ...arg_n; /* Teilketten */
```

SEMANTIK:

strcat kopiert eine Reihe von Zeichenketten in eine Zielzeichenkette ds. Jede der einzelnen Zeichenketten wird durch NUL ('\0') abgeschlossen. Die einzelnen Zeichenketten werden in der Reihenfolge von links nach rechts kopiert, wobei auf das letzte Zeichen einer Kette sofort das erste Zeichen der naechsten Kette folgt (d.h. das Zeichenkettenende wird jeweils ignoriert).
Als letztes Argument in der Funktion wird das Zeichen NUL angegeben. Damit wird die Zielzeichenkette abgeschlossen und die Kopieroperation beendet.

ERGEBNIS:
strcat liefert die Position des Kettenendezeichens.

BEISPIEL:
Verketten der Zeichenketten "#include" und "<TEST.C>"

```
char *line;
...
strcat (line, "#include", "<TEST.C>", NULL);
```

18.14. strncmp - Vergleichen zweier Puffer

PARAMETER:
```
int strncmp (s1, s2, n)
   char *s1, *s2;  /* Puffer */
   int n;          /* Anzahl der Zeichen */
```

SEMANTIK:
strncmp vergleicht n Zeichen der zwei Textpuffer s1 und s2 zeichenweise miteinander.

ERGEBNIS:
strncmp liefert eine ganze Zahl groesser, gleich oder kleiner Null, jenachdem s1 lexikographisch groesser, gleich oder kleiner s2 ist.

18.15. strcmp - Vergleich zweier Zeichenketten

PARAMETER:
```
int strcmp (s1, s2)
   char *s1, *s2;  /* Zeichenketten */
```

SEMANTIK:
strcmp vergleicht zwei Zeichenketten s1 und s2 zeichenweise, einschliesslich Endezeichen ('\0'), miteinander.

ERGEBNIS:
strcmp liefert einen ganzzahligen Wert groesser, gleich oder kleiner Null, jenachdem, ob s1 lexikographisch groesser, gleich oder kleiner s2 ist.

18.16. prefix - Pruefen, ob eine Zeichenkette Vorsatz einer anderen ist

PARAMETER:
```
int prefix (s1, s2)
   char *s1, *s2;  /* Zeichenketten */
```

SEMANTIK:
prefix vergleicht zwei Zeichenketten s1 und s2 zeichenweise, bis ausschliesslich Endezeichen ('\0'), miteinander. Im Unterschied zur Funktion strcmp() kann die erste Zeichenkette laenger sein als die zweite.

ERGEBNIS:
prefix liefert den Wahrheitswert WAHR (=1), wenn die zweite Zeichenkette s2 Vorsatz der ersten Zeichenkette s1 ist, sonst FALSCH (=0).

BEISPIEL:
Einfuegen einer Datei, wenn eine Zeile "line" mit den Zeichen "#include" beginnt

```
char *line;
...
if (prefix (line, "#include")
... Einfuegen Datei ...
```

18.17. index/rindex - Suchen eines Zeichens in einem Puffer

PARAMETER:

```
int index (s, n, c)
   char *s;          /* Puffer */
   int n;            /* Pufferlaenge */
   char c;           /* Textzeichen */

int rindex (s, n, c)
   char *s;
   int n;
   char c;
```

SEMANTIK:
index bzw. rindex durchsucht einen Puffer s der Laenge n nach dem ersten Auftreten des vorgegebenen Zeichens c.

ERGEBNIS:
index liefert die erste und rindex die letzte Position des Zeichens im Puffer oder die Pufferlaenge, falls das Zeichen nicht auftritt.

BEISPIEL:
Suchen des Textzeichens "#" und pruefen, ob es sich in der 1.Position einer Textzeile befindet.

```
char *line;
if (index (line, 80, '#') == 1)
```

18.18. substr - Suchen einer Zeichenkette

PARAMETER:

```
int substr (s1, n1, s2, n2)
   char *s1, *s2;    /* Zeichenkette */
   int n1, n2;       /* Zeichenanzahl */
```

SEMANTIK:
substr zerlegt die Kette s1 der Laenge n1 und sucht dabei die Zeichenkette s2 der Laenge n2. Nach dem ersten Auftreten der Zeichenkette wird die Funktion verlassen.

ERGEBNIS:
substr liefert die Position der Kette s2 innerhalb von s1 oder die Laenge n1 falls die Zeichenkette nicht auftritt.

BEISPIEL:
Suchen einer Zeichenkette "*.*" in einer Textzeile der Laenge 80 Byte.

```
char *line;
...
if (substr (line, 80, "*.*", 3) < 80)
```

18.18. instr - Suchen verschiedener Zeichen

PARAMETER:
```
int instr (s, f, n)
   char *s,          /* Puffer */
        *f;          /* Zeichenfolge */
   int n;            /* Laenge */
```

SEMANTIK:
instr durchsucht einen Puffer s der Laenge n nach dem ersten Auftreten eines Zeichens aus einer Zeichenfolge f, die durch NUL ('\Ø') begrenzt wird.
Soll das Zeichen NUL selbst Bestandteil der Folge sein, so muss es an erster Stelle stehen.

ERGEBNIS:
instr liefert entweder die Position des ersten Zeichens, welches in der Zeichenfolge vorkommt, oder aber die Pufferlaenge, falls keines der Zeichen vorhanden ist.

BEISPIEL:
Ersetzen aller FF, CR und TAB in einem 512-Byte-Puffer durch Leerzeichen.

```
char *buf; int i, n;
...
while ((s = instr (buf, n, "\n\f\t")) < n)
        buf[s] = ' ';
...
```

18.19. notstr - Suchen bestimmter Zeichen

PARAMETER:
```
int notstr  (s, f, n)
   char *s,          /* Puffer */
        *f;          /* Zeichenfolge */
   int n;            /* Laenge */
```

SEMANTIK:
notstr durchsucht einen Puffer s der Laenge n nach dem ersten Auftreten eines Zeichens, welches nicht in einer durch NUL ('\Ø') begrenzten Zeichenfolge f vorkommt.

Soll das Zeichen NUL selbst Bestandteil der Folge sein, so muss es an erster Stelle stehen.

ERGEBNIS:

notstr liefert entweder die Position des ersten Zeichens, welches nicht in der Zeichenfolge vorkommt, oder die Pufferlaenge, falls alle Zeichen auch in der Folge enthalten sind.

BEISPIEL:

Durchsuchen eines Zahlenpuffers auf erlaubte Zeichen (Dezimalziffern).

```
char *buf; int n;
...
if (notstr (buf, n, "0123456789") != n)
      errfmt("illegal number\n");
...
```

19. Konvertierfunktionen

Die nachfolgenden Funktionen dienen insbesondere der Kommunikation des Bedieners mit dem Programm. Daten koennen von ihrer internen Darstellung in eine lesbare Form gewandelt werden und umgekehrt.

19.1. atod - Konvertieren eines Textpuffers in eine Gleitkommazahl doppelter Genauigkeit

PARAMETER:

```
int atod (s, n, pdnum)
   char *s;          /* Zeichenkette */
   int n;            /* Zeichenanzahl */
   double *pdnum;    /* Gleitkommazahl */
```

SEMANTIK:

atod konvertiert die Zeichenkette s, bestehend aus n Zeichen, in eine Gleitkommazahl doppelter Genauigkeit.
Die Zeichenkette beinhaltet die Textdarstellung einer Gleitkommazahl entsprechend der vorgegebenen Syntax.
Fuehrende Leerzeichen werden uebergangen und ein Vorzeichen ist zugelassen.
Die Konvertierung wird entweder bei Auftreten eines unzulaessigen Zeichens oder am Ende der Zeichenkette beendet.

Eine zulaessige Eingabe hat die Form:

```
[+|-]d[.d][e|E[+|-]d]
(d... ganze Zahl in Dezimaldarstellung)
```

Beispiel:
+1.2e-1 oder -3E2 ...

Beachte:
Es wird nicht auf Ueber- oder Unterschreitung des zulaessigen Wertebereiches kontrolliert.

ERGEBNIS:
atod liefert die Anzahl der verarbeiteten Zeichen, also einen Wert zwischen 0 und n.
Die konvertierte Zahl wird in der Variablen **pdnum** abgelegt.

19.2. dtoe - Konvertieren einer Gleitkommazahl in einen Textpuffer unter Verwendung des e-Formates

PARAMETER:
```
int dtoe (s, dbl, p, q)
   char *s;          /* Zeichenkette */
   double dbl;       /* Gleitkommazahl */
   int p,            /* Anzahl der Dezimalziffern links */
       q;            /* bzw. rechts vom Dezimalpunkt */
```

SEMANTIK:
dtoe konvertiert die Gleitkommazahl doppelter Genauigkeit **dbl** in eine Zeichenkette **s**.
Dabei spezifieziert der Parameter **p** die Anzahl der Dezimalziffern vor und **q** die nach dem Dezimalpunkt.

Die erzeugte Textdarstellung hat die Form:

```
[-]d.d e|E +|- d
(d... ganze Zahl in Dezimaldarstellung)
```

ERGEBNIS:
dtoe liefert die Anzahl der zur Darstellung der Gleitkommazahl verarbeiteten Zeichen.

19.3. dtof - Konvertieren einer Gleitkommazahl in einen Textpuffer unter Verwendung des f-Formates

PARAMETER:
```
int dtof (s, dbl, p, q)
   char *s;          /* Zeichenkette */
   double dbl;       /* Gleitkommazahl */
   int p,            /* Anzahl der Dezimalziffern links */
       q;            /* bzw. rechts vom Dezimalpunkt */
```

SEMANTIK:
dtof konvertiert die Gleitkommazahl doppelter Genauigkeit **dbl** in eine Zeichenkette **s**).
Dabei spezifiziert der Parameter **p** die Anzahl der Dezimalziffern vor und **q** die nach dem Dezimalpunkt.

Die erzeugte Textdarstellung hat die Form:

```
[-]d.d      (d... ganze Zahl in Dezimaldarstellung )
```

ERGEBNIS:
dtof liefert die Anzahl der zur Darstellung der Gleitkommazahl verarbeiteten Zeichen.

19.4. atoi - Konvertieren eines Textpuffers in eine ganze Zahl

PARAMETER:
```
int atoi (s, n, pinum, base)
   char *s;          /* Zeichenkette */
   int n,            /* Zeichenanzahl */
         pinum,      /* ganze Zahl */
         base;       /* Basis */
```

SEMANTIK:
atoi konvertiert die Zeichenkette **s**, bestehend aus **n** Zeichen, in eine ganze Zahl und legt diese in der Variablen **pinum** ab.
Die Zeichenkette beinhaltet die Textdarstellung einer ganzen Zahl zur Basis 10 (dezimal), 8 (oktal) oder 16 (hexadezimal), entsprechend der Syntaxregeln. Fuehrende Leerzeichen und ein fuehrendes '0x' bzw. '0X' bei Hexadarstellung wird uebergangen. Ein Vorzeichen ist zugelassen. Die Konvertierung wird am Ende der Zeichenkette oder bei Auftreten des ersten unzulaessigen Zeichens beendet.
Beachte:
Es wird nicht auf Ueber- oder Unterschreitung des Wertebereiches oder auf unzulaessige Basiswerte kontrolliert.

ERGEBNIS:
atoi liefert die Anzahl der tatsaechlich verarbeiteten Zeichen, also einen Wert zwischen 0 und n. Die konvertierte Zahl wird in der Variablen **pinum** abgelegt.

19.5. itoa - Konvertieren einer ganzen Zahl in einen Textpuffer

PARAMETER:
```
int itoa (s, i, base)
     char *s;          /* Zeichenkette */
     int i,            /* ganze Zahl */
         base;         /* Basis */
```

SEMANTIK:
itoa konvertiert die ganze Zahl **i** zur Basis **base** in eine Zeichenkette **s**.
Wenn die Basis positiv (**base** > 0) ist, wird die ganze Zahl als vorzeichenlose Zahl behandelt; ist die Basis negativ (**base** < 0), wird die ganze Zahl als negative Zahl zur Basis **-base** betrachtet. Fuer **base** == 0 wird zur Basis -10 konvertiert.

Beachte:
Es wird nicht auf Zulaessigkeit geprueft.

ERGEBNIS:

itoa liefert die Anzahl der Zeichen zur Darstellung der ganzen Zahl, die aus 4 bis 8 Ziffern und einem Vorzeichen bestehen kann.

19.6. atol - Konvertieren eines Textpuffers in eine lange ganze Zahl

PARAMETER:

```
long atol(s, n, pinum, base)
   char *s;          /* Zeichenkette */
   int n,            /* Zeichenanzahl */
   long pinum;       /* lange ganze Zahl */
   int base;         /* Basis */
```

SEMANTIK:

atol konvertiert die Zeichenkette s, bestehend aus n Zeichen, in eine lange ganze Zahl und legt diese in der Variablen pinum ab.
Die Zeichenkette beinhaltet die Textdarstellung einer ganzen Zahl zur Basis 10 (dezimal), 8 (oktal) oder 16 (hexadezimal), entsprechend der Syntaxregeln. Fuehrende Leerzeichen und ein fuehrendes '0x' bzw. '0X' bei Hexadarstellung wird uebergangen. Ein Vorzeichen ist zugelassen. Die Konvertierung wird am Ende der Zeichenkette oder bei Auftreten des ersten unzulaessigen Zeichens beendet.

Beachte:
Es wird nicht auf Ueber- oder Unterschreitung des Wertebereiches oder auf unzulaessige Basiswerte kontrolliert.

ERGEBNIS:

atol liefert die Anzahl der tatsaechlich verarbeiteten Zeichen, also einen Wert zwischen 0 und n.
Die konvertierte Zahl wird in der Variablen pinum abgelegt.

19.7. ltoa - Konvertieren einer langen ganzen Zahl in einen Textpuffer

PARAMETER:

```
int ltob (s, i, base)
   char *s;          /* Zeichenkette */
   long i;           /* lange ganze Zahl */
   int   base;       /* Basis */
```

SEMANTIK:

ltoa konvertiert die lange ganze Zahl i zur Basis base in eine Zeichenkette s.
Wenn die Basis positiv (base > 0) ist, wird die ganze Zahl als vorzeichenlose Zahl behandelt; ist die Basis negativ (base < 0), wird die ganze Zahl als negative Zahl zur Basis -base betrachtet. Fuer base == 0 wird zur Basis -10 konvertiert.

ERGEBNIS:
ltoa liefert die Anzahl der Zeichen zur Darstellung der ganzen Zahl, die aus 4 bis 8 Ziffern und einem Vorzeichen bestehen kann.

20. Arithmetische Funktionen

20.1. abs - Finden des absoluten Betrages

PARAMETER:
abs (a)
(die Variable a kann jeden numerischen Wert annehmen)

SEMANTIK:
abs ermittelt den absoluten Betrag des in der Funktion uebergebenen Parameters a.

ERGEBNIS:
abs liefert den ermittelten numerischen Wert.

20.2. max - Test auf Maximum

PARAMETER:
max (a, b)
(die Variablen a und b koennen jeden numerischen Wert annehmen)

SEMANTIK:
max ermittelt das Maximum der in der Funktion uebergebenen Parameter.
Die Parameter a und b koennen dabei jeden numerischen Wert annehmen, die Typumwandlung erfolgt automatisch.

ERGEBNIS:
max liefert den ermittelten numerischen Wert.

20.3. min - Test auf Minimum

PARAMETER:
min (a,b)
(die Variablen a und b koennen jeden numerischen Wert annehmen)

SEMANTIK:
min ermittelt das Minimum der in der Funktion uebergebenen Parameter.
Die Parameter koennen dabei jeden beliebigen numerischen Wert annehmen, die Typumwandlung erfolgt automatisch.

ERGEBNIS:
min liefert den ermittelten numerischen Wert.

20.4. uldiv - Division vorzeichenloser langer Integerzahlen

PARAMETER:

```
long uldiv (n, d)
   long n, d;        /* Dividend, Divisor */
```

SEMANTIK:

uldiv dividiert die lange Integerzahl n durch die lange Integerzahl d, wobei beide ohne Vorzeichen betrachtet werden.

ERGEBNIS:

uldiv liefert den Quotient aus der Division von n/d.

20.5. ulmod - Restdivision vorzeichenloser langer Integerzahlen

PARAMETER:

```
long ulmod (n, d)
   long n, d;        /* Dividend, Divisor */
```

SEMANTIK:

ulmod dividiert die lange Integerzahl n durch die lange Integerzahl d, wobei beide ohne Vorzeichen betrachtet werden.

ERGEBNIS:

ulmod liefert den ganzzahligen Rest aus der Division von n/d.

21. Arbeit mit dem 1/2"-Magnetband

Die Arbeit mit dem 1/2"-Magnetband wird durch spezielle Funktionen, die in einer Bibliothek zusammengefasst sind, realisiert.
Zusaetzlich zu den im Kapitel 1 der Anwenderdokumentation zum C-Compiler angefuehrten Dateien muessen auf Diskette folgende Dateien vorhanden sein:

MBG.REL	physische Ansteuerung Magnetband
MBLIB.REL	Bibliothek der Magnetbandfunktionen
MB.SUB	Kommandodatei fuer Magnetbandarbeit

Die Uebersetzung eines C-Programmes mit Magnetbandfunktionen wird somit durch das Kommando:

A) SUBM MB <d:dateiname>[P]

analog der ueblichen Kommandoform realisiert.

Die Bibliothek **MBLIB** enthaelt Funktionen zum Positionieren, Lesen und Schreiben.

21.1. mbpos - Positionieren Magnetband

PARAMETER:

```
mbpos (fctc, cn, seb)
   int fctc;    /* Funktionskode */
   int cn;      /* Anzahl */
   int seb;     /* Adresse Statusinformation */
```

SEMANTIK:

mbpos positioniert das Magnetband entsprechend dem angegebenen Funktionskode fctc und der Anzahl cn.

fctc	Bedeutung		cn
0x04	WTM	Aufzeichnen zweier Bandmarken (BM) und positionieren zwischen diese BM	0 (ohne Bedeutung)
0x08	FSR	Vorwaertspositionieren ueber cn Bloecke	1 - 32767
0x88	BSR	Rueckwaertspositionieren ueber cn Bloecke	
0x10	FSF	Vorwaertspositionieren ueber cn BM	
0x90	BSF	Rueckwaertspositionieren ueber cn BM und Vorsetzen ueber letzte BM	
0x20	REW	Rueckspulen bis Bandanfangsmarke (BOT)	0 (ohne Bedeutung)

Die Positionierfunktion wird vor Realisierung der vorgegebenen Anzahl cn Bloecke bzw. BM beendet, wenn

a) - beim Rueckpositionieren BOT erreicht wird

b) - bei "Vorwaertspositionieren ueber cn BM" das logische Bandende (EOF) erreicht wird
   - bei "Vorwaertspositionieren ueber cn Bloecke" oder "Rueckwartspositionieren ueber cn Bloecke" ueber eine BM positioniert wird

ERGEBNIS:

mbpos liefert als Ergebnis die echt realisierte Anzahl (positionierte Bloecke bzw. BM) bei korrekter Ausfuehrung der Funktion oder den Wert -1 bei Fehlern.

Eine zusaetzliche Information (Statusinformation) ueber die Ausfuehrung bzw. Fehlerursache wird in dem Speicherbereich als Integer-Zahl abgelegt, dessen Adresse als Parameter **seb** angegeben ist.

Allgemein, d.h. fuer alle Magnetbandfunktionen, gilt fuer die Statusinformation folgende Bitbelegung:

	Bit	Bedeutung (Bit = 1)
	0	MB nicht bereit
	1	0
	2	vorgegebene Anzahl (Laenge) ≠ realisierte Anzahl (Laenge)
L	3	Bandanfang (BOT)
	4	Bandende (EOT)
	5	Bandmarke (BM)
	6	Schreibring fehlt
	7	Schreib-/Lese-/Geraetefehler
H	8 - 15	0

(L ... niederwertiger Teil, H... hoeherwertiger Teil der Integer-Zahl)

Als fehlerhaft (Funktionswert = -1) wird die Ausfuehrung einer Magnetbandfunktion bewertet, wenn eines der Bits 0, 6 oder 7 den Wert 1 hat.

BEISPIEL:
Rueckspulen des Magnetbandes an den Bandanfang und Ausgabe einer Fehlernachricht, falls das Magnetband nicht bereit ist.

```
#define REW 0x20
    ...
    int status;
    ...
    if (mbpos (REW, 0, &status) < 0 && status&0x01 == 0x01)
        errfmt("MB nicht bereit\n");
    ...
```

21.2. mbread - Lesen Block

PARAMETER:

```
mbread (ein, n, seb)
   char *ein;    /* Einlesebereich */
   int n;        /* Laenge Einlesebereich */
   int *seb;     /* Adresse Statusinformation */
```

SEMANTIK:

mbread liest einen Block vom Magnetband in einen Einlesebereich ein der Laenge **n**.
Beim Lesen einer BM wird vor diese BM positioniert.

ERGEBNIS:

mbread liefert die echte Laenge des Bereiches bzw. den Wert -1 bei fehlerhafter Ausfuehrung.
Ist die Laenge des gelesenen Blockes groesser als die Laenge des Eingabebereiches, werden die darueberhinausgehenden Stellen abgeschnitten.
Ist die Blocklaenge kleiner als die des Eingabebereiches, bleiben die restlichen Stellen des Eingabebereiches unveraendert.

Ein Lesefehler wird nach 8 fehlerhaften Leseversuchen erkannt, dann wird vor den Block positioniert.

Entsprechend der Bitbelegung der Statusinformation wird ein Kode in den durch **seb** adressierten Speicherbereich abgelegt, z.B.:

0x00 fehlerfreie Ausfuehrung:
Laenge Eingabebereich gleich Blocklaenge

0x04 fehlerfreie Ausfuehrung:
Laenge Eingabebereich ungleich Blocklaenge

0x14 fehlerfreie Ausfuehrung:
EOT ueberlesen

0x24 fehlerfreie Ausfuehrung:
BM gelesen

0x80 Lese-/Geraetefehler, Blockluecke zu gross

BEISPIEL:

Lesen eines Blockes in einen Eingabebereich der Laenge 4096 Byte und Anzeige der echten Blocklaenge bei fehlerfreier Ausfuehrung:

```
int l, status;
char ein[4096];
...
if ((l = mbread (ein, 4096, &status)) != -1)
    printf("Blocklaenge = %i\n", l);
...
```

21.3. mbwrite - Schreiben Block

PARAMETER:

```
mbwrite (aus, n, seb)
   char *aus;    /* Ausgabebereich */
   int n;        /* Anzahl auszugebende Zeichen */
   int *seb;     /* Adresse Statusinformation */
```

SEMANTIK:

mbwrite schreibt einen Block der Laenge n (2 <= n und n <= 16383) aus einem Ausgabebereich aus auf Magnetband.

ERGEBNIS:

mbwrite liefert den Wert 0 oder -1 bei fehlerhafter Ausfuehrung.

Ein Schreibfehler wird nach je 4 fehlerhaften Aufzeichnungsversuchen auf 4 aufeinanderfolgende Bandstellen erkannt und das Magnetband vor den fehlerhaft aufgezeichneten Block positioniert.

Entsprechend der Bitbelegung der Statusinformation wird z.B. als Kode in den durch **seb** adressierten Speicherbereich abgelegt:

0x00 fehlerfreie Ausfuehrung

0x10 beim Aufzeichnen EOT ueberschrieben

0x40 Schreibring fehlt bzw. keine Spule eingelegt

0x80 Schreib-/Geraetefehler

BEISPIEL:

Schreiben der ersten 64 Zeichen eines Bereiches, Ausgabe als Block auf Magnetband und Anzeige bei Fehler:

```
char ausgabe[MAXL];
int status;
...
if ((mbwrite (aus, 64, &status) == -1)
    errfmt ("Schreibfehler\n");
...
```

Weitere Ausfuehrungen zur Magnetbandarbeit sind in "Zusatzsoftware II" im Kapitel "Treiber fuer 1/2"-Magnetbandgeraet MBG 1520 (SCPX)" enthalten.

Zusammenfassung der Bibliotheksfunktionen

1. Programmsteuerung

main()	- logischer Programmanfang	(S)
exit()	- Verlassen eines C-Programmes	(S)
getflags()	- Auswertung Kommandozeile	(E)

2. Standard-E/A

scanf()	- formatgesteuerte Eingabe	(S)
printf()	- formatgesteuerte Ausgabe	(S)
errfmt()	- formatgesteuerte Fehlerausgabe	(E)
getchar()	- Zeicheneingabe	(S)
putchar()	- Zeichenausgabe	(S)
getlin()	- Zeileneingabe	(E)
putlin()	- Zeilenausgabe	(E)

3. Dateiarbeit

finit()	- Initialisieren E/A-Struktur	(E)
fcreate()	- Erzeugen Datei	(E)
fopen()	- Eroeffnen Datei	(S)
fclose()	- Schliessen Datei	(S)
fscanf()	- formatgesteuerte Eingabe	(S)
fprintf()	- formatgesteuerte Ausgabe	(S)
getc()	- Zeicheneingabe	(S)
putc()	- Zeichenausgabe	(S)
getl()	- Zeileneingabe	(E)
putl()	- Zeilenausgabe	(E)

4. Systemschnittstelle

create()	- Erzeugen Datei	(S)
open()	- Eroeffnen Datei	(S)
close()	- Schliessen Datei	(S)
read()	- Physisches Lesen	(S)
write()	- Physisches Schreiben	(S)
lseek()	- Direkt Positionieren	(S)
unlink()	- Loeschen Datei	(S)
uname()	- Erzeugen universellen Dateinamen	(E)
scp()	- Zugriff auf BDOS	(E)
in()	- Eingabe von einem Port	(E)
out()	- Ausgabe auf ein Port	(E)

5. Dynamische Speicherplatzverwaltung

alloc()	- Zuweisen von Speicherplatz	(S)
buybuf()	- Zuweisen von Speicherplatz und Kopieren Puffer	(E)
free()	- Freigeben von Speicherplatz	(S)

6. Verarbeiten von Zeichen und Zeichenketten

iswhite()	-)	
isdigit()	-)	
isalpha()	- } Zeichenanalyse	(S)
isupper()	-)	
islower()	-)	
tolower()	- Wandlung in Kleinbuchstaben	(S)
toupper()	- Wandlung in Grossbuchstaben	(S)

6. Verarbeiten von Zeichen und Zeichenketten

strlen()	- Laengenberechnung	(S)
fill()	- Initialisieren Textpuffer	(E)
squeeze()	- Verdichten Textpuffer	(E)
strncpy()	- Kopieren Puffer	(S)
strcpy()	- Kopieren Zeichenkette	(S)
strcat()	- Verketten von Zeichenketten	(E)
strncmp()	- Vergleichen Puffer	(S)
strcmp()	- Vergleichen Zeichenketten	(S)
prefix()	- Vergleichen Teilketten	(E)
index()	- Suchen eines Zeichens	(S)
rindex()	- Suchen eines Zeichens	(S)
substr()	- Suchen Teilzeichenkette	(E)
instr()	- Suchen verschiedener Zeichen	(E)
notstr()	- Suchen bestimmter Zeichen	(E)

7. Konvertierung

atod()	- Text in Gleitkommazahl	(S)
dtoe()	- Gleitkommazahl in Text (e-Format)	(S)
dtof()	- Gleitkommazahl in Text (f-Format)	(S)
atoi()	- Text in ganze Zahl	(S)
itoa()	- Ganze Zahl in Text	(S)
atol()	- Text in lange ganze Zahl	(S)
ltoa()	- Lange ganze Zahl in Text	(S)

8. Arithmetische Funktionen

abs()	- Absoluter Betrag	(S)
max()	- Maximum	(S)
min()	- Minimum	(S)

8. Arithmetische Funktionen

uldiv()	- Division langer Integerzahlen	(E)
ulmod()	- Restdivision langer Integerzahlen	(E)

9. Magnetbandfunktionen

mbpos()	- Positioneieren	(E)
mbread()	- Lesen	(E)
mbwrite()	- Schreiben	(E)

(S) ... entspricht Vorbildtyp
(E) ... Erweiterung

5. Routine fuer schluesselindizierten Zugriff KISAMC 1520 (SCPX)

5.1. Charakteristik und Leistungsumfang

KISAMC ist ein eigenstaendiges Dateiverarbeitungssystem fuer schluesselindizierten Zugriff, bereitgestellt als Unterprogramm zum Einbinden in SCP-Anwendersoftware.

Es steht als REL-Datei zur Verfuegung und kann zu allen Objektdateien von Compilern bzw. vom Assembler gebunden werden.

KISAMC realisiert die Dateifunktionen:

OPEN	- Eroeffnen Datei und Aufbau Indextabelle
CLOSE	- Abschliessen Datei
READ	- Lesen ab aktueller Dateiposition
KEY	- schluesselindizierter Zugriff
EXTEND	- Schreiben bzw. Erweitern Datei
UPDATE	- Rueckschreiben eines Satzes

- Hauptspeicherumfang: 3,5 kBytes

- KISAMC erstellt Dateien in lueckenloser Satzfolge und laesst beliebige feste logische Satzlaengen zu. Jede Datei wird von KISAMC kompatibel zum Sortierprogramm SORT 1520 (SCPX) abgeschlossen :

  2 Bytes FFH und Auffuellen des Restes im letzten Record mit 1A-Zeichen.

  Hinweis:
  Belegt die Datei den letzten Record vollstaendig, faellt diese Endekennung weg.

- KISAMC-Dateien entsprechen den Erfordernissen des Sortierprogrammes SORT 1520 (SCPX) und koennen mit diesem verarbeitet werden.

- Die **mittlere Zugriffszeit** auf einen logischen Datensatz ueber die KEY-Funktion betraegt ca. 0,3 sec. (bei einer Segmentierung von 10 Seatzen je Segment). Sie ist von der Segmentierung der Datei abhaengig.

- **Aufruf und Schnittstelle zum Anwenderprogramm:**
  Der Aufruf von KISAMC erfolgt als Unterprogramm- bzw. Funktionsaufruf. Der Eintrittspunkt in KISAMC haengt von der Programmiersprache des aufrufenden Programmes ab.

  Eintrittspunkte:

KISAMC	- fuer Assembler-, FORTRAN- und PASCAL-Programme
CKISAM	- fuer C-Programme
BKISAM	- fuer BASIC-Programme

Die Art der Parameterbereitstellung haengt von der Programmiersprache des aufrufenden Programmes ab und wird im Anhang speziell erlaeutert.

In Abhaengkeit von der auszufuehrenden Funktion verlangt KISAMC ein bis drei Parameter. Der erste Parameter ist fuer alle Funktionsaufrufe gleich. Er wird als Zeichenkette gefordert und beinhaltet:

Byte 0: Funktionscode fuer die auszufuehrende Funktion

01H = OPEN - Funktion
02H = KEY - Funktion
03H = UPDATE - Funktion
04H = EXTEND - Funktion
05H = READ - Funktion
06H = CLOSE - Funktion

Byte 1 u. 2: Laufwerksbezeichnung einschliesslich Doppelpunkt

- zugelassen sind die Bezeichnungen A...E (Gross- oder Kleinbuchstaben)

- Doppelpunkt ist zwingend und entspricht dem Dialogkonzept im SCP

Byte 3 bis 14: Dateiname und Dateityp (Dateibezeichnung)

- Die Angabe des Dateinamens und -typs ist identisch zum SCPX-Dialogkonzept und muss wie folgt notiert sein:

  . maximal 8 Stellen Dateiname
  . Punkt (".")
  . 3 Stellen Dateityp

- Fuer Dateiname und Dateityp sind alle alphanumerischen Zeichen und Sonderzeichen verwendbar, ausser

  <|>|.|,|;|:|=|?|*|[|]

  Die Wandlung in Grossbüchstaben wird von KISAMC ausgefuehrt.

Alle weiteren, funktionsspezifischen Parameter werden nachfolgend in der Beschreibung der einzelnen Funktionen erlaeutert.
Generell verlangt KISAMC eine stellungsabhaengige Parameteruebergabe ueber die CPU-Register HL, DE, BC:

- Register HL : Parameter 1 (Adresszeiger auf Funktionscode, Laufwerksbezeichnung, Dateibezeichnung)
- Register DE : Parameter 2 (Adresszeiger auf funktionsabhaengigen Parameter)
- Register BC : Parameter 3 (Adresszeiger auf funktionsabhaengigen Parameter)

In FORTRAN- und BASIC-Programmen wird bei Aufruf eines Unterprogrammes mit Parameteruebergabe das Laden der CPU-Register in der angegebenen Weise automatisch generiert. Fuer C-Programme wurde in KISAMC ein entsprechender Uebergang geschaffen, der das Laden der CPU-Register realisiert.

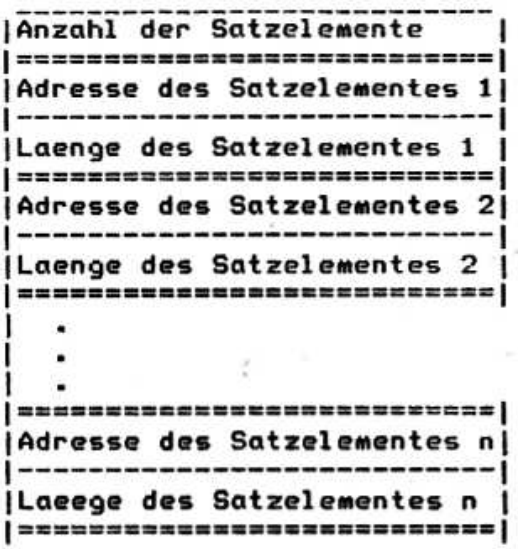
- **Rueckkehrcode:**
KISAMC gibt an das rufende Programm im Byte 0 des ersten Parameters (bei Aufruf Funktionscode) einen Rueckkehrcode zurueck. Er gibt Auskunft darueber, ob die angesteuerte Funktion ausgefuehrt werden konnte oder welche Fehler dabei auftraten, ob Fehlermassnahmen von KISAMC selbst vorgenommen wurden, oder der Rueckkehrcode enthaelt lediglich eine Nachricht. Die Nachricht beschreibt die Faelle EOF, leere Datei bzw. Datei neu angelegt.
Nach der Rueckkehr in das rufende Programm sind die fuer jede Funktion spezifischen oder allgemeinen Rueckkehrcodes zu testen und die entsprechenden Massnahmen im Programm, falls erforderlich, einzuleiten. Bei schwerwiegenden Fehlern nimmt KISAMC eine Dateisicherung vor, indem es die Datei auf dem aktuellen Stand abschliesst.
Eine Uebersicht ueber die Rueckkehrcodes, ihre Bedeutung und die erforderlichen Massnahmen im rufenden Programm werden im Abschnitt 5.4. gegeben.

5.2. Anwendungsbedingungen

1. KISAMC verlangt eine nach Schluesselbegriffen binear aufsteigend sortierte Datei fuer die KEY-Funktion.

2. Der Schluesselbegriff darf nur ASCII-Zeichen mit Byte-Werten zwischen 20H und 7FH enthalten.
Die maximale Laenge eines Schluesselbegriffes ist 255.
KISAMC verarbeitet nur zusammenhaengende Schluesselbegriffe. Der Schluesselbegriff kann innerhalb des Datensatzes an beliebiger Position stehen.

3. Die Datensatzlaenge (logische Satzlaenge) kann beliebig sein. Innerhalb einer Datei wird eine feste logische Satzlaenge verlangt.
Die Datensaetze koennen beliebige Daten enthalten (00H bis FFH), ausgenommen der Schluesselbegriff. KISAMC uebergibt und schreibt Daten in der Form, wie sie vom aufrufenden Programm uebergeben wurden. Die Elemente eines Datensatzes werden vom Anwender selbst bestimmt und verwaltet. KISAMC verlangt beim Lesen bzw. Schreiben eines Datensatzes die Uebergabe eines Vektors (INTEGER-Feld), der den Aufbau des Datensatzes bezueglich seiner Satzelemente beschreibt:



Diese Art der Parameteruebergabe sichert im BASIC und FORTRAN die Moeglichkeit, einen zusammenhaengenden Datensatz aus Elementen verschiedenen Typs bereitzustellen. Da in PASCAL die RECORD- und in der C-Sprache die STRUKTUR-Vereinbarung die Anordnung eines zusammenhaengenden Datensatzes erlaubt, kann fuer diese Programme die Anzahl der Datensatzelemente zu 1 gesetzt und nachfolgend im zweiten und dritten Element des Vektors die Anfangsadresse und die Laenge des Datensatzes notiert werden.

4. KISAMC verarbeitet Dateien, die nicht mit ihm erstellt wurden. Voraussetzung ist, dass sie eine feste logische Satzlaenge besitzen und der Rest des letzten physischen Records mit 2 Zeichen FFH und 1AH-Zeichen (siehe Abschnitt 5.1.) aufgefuellt ist. Es ist zu beachten, dass eventuelle Satztrennzeichen in die logische Satzlaenge einzubeziehen sind.

   Hinweis:
   Die Endekennung kann mit dem Programm DIENST 1520 (SCPX) an die Datei angefuegt werden.

5. Mittels KISAMC bearbeitete und erstellte Dateien sind von dem Sortierprogramm SORT 1520 (SCPX) verarbeitbar. Die Eingabedatei (zu sortierende Datei) und die Ausgabedatei (sortierte Datei) muessen folgende Attribute haben (siehe SORT-Dokumentation, "Zusatzsoftware II"):

   ```
   FIXED-LENGTH
   FFZZZ
   ```

6. KISAMC kann maximal 10 Dateien gleichzeitig bearbeiten, d.h., maximal 10 Dateien koennen gleichzeitig eroeffnet sein. Mit dem Aufruf der CLOSE-Funktion in KISAMC wird die jeweilige Eintragung der Datei in KISAMC geloescht. Damit wird in KISAMC Platz fuer eine neue Dateieintragung frei

und ist fuer eine neu zu eroeffnende Datei wieder nutzbar.

7. Jede eroeffnete schluesselindizierte Datei erfordert einen Bereich fuer die Indextabelle. Der Platz fuer die Indextabelle(n) muss vom Anwenderprogramm bereitgestellt werden. KISAMC uebernimmt beim Eroeffnen einer schluesselindizierten Datei die Anfangsadresse des im Anwenderprogramm fuer die Indextabelle definierten Bereiches.
Fuer die Indextabelle(n) sollte maximaler Speicherraum bereitgestellt werden, um eine starke Segmentierung der schluesselindizierten Dateien zu ermoeglichen. Damit sind minimimale Zugriffszeiten garantiert.
Der Anwender hat sich vor der Definition der Bereiche fuer die Indextabelle(n) ueber die Groesse des Restspeichers zu informieren, um diesen fuer die Indextabelle(n) voll nutzbar zu machen. Das heisst, der Speicherraum der Indextabelle(n) sollte so gross wie der gesamte moegliche Restspeicher definiert sein.

Hinweis:
Wird waehrend der Laufzeit des Anwenderprogrammes eine Datei abgeschlossen (CLOSE-Funktion), wird der fuer die Datei reservierte Bereich fuer die Indextabelle frei und kann fuer eine nachfolgend zu eroeffnende Datei genutzt werden. Diese Arbeitsweise ist speicheroptimal und unbedingt zu nutzen.

5.3. Datei-Funktionen

5.3.1. OPEN-Funktion

1. Aufgabe

- Uebernahme und Pruefung der Datei-Parameter
- Generieren FCB fuer die entsprechende Datei
- Eroeffnen der Datei bzw. Anlegen einer neuen Datei, falls noch keine Dateieintragung unter dem angegebenen Dateinamen existiert
- Aufbau der Indextabelle, falls die eroeffnete Datei existiert bzw. keine leere Datei ist
- Aufbau eines Datei-Informationsblockes fuer die Datei, der alle Dateispezifikationen sowie alle wichtigen Dateipositionen enthaelt (Schluessellaenge, Schluesselposition, logische Satzlaenge, Laenge Indextabelle, aktuelle Dateiposition, Dateiendeposition)
- Sortierpruefung (Die Sortierpruefung bezieht sich nur auf alle in der Indextabelle eingetragenen Schluessel.)

Bemerkungen:

1) FCB-Bereich, Datei-Informatinsblock und Puffer-Bereich fuer einen logischen Datensatz belegen den Anfangsbereich der reservierten Indextabelle. Es ergibt sich folgende Belegung:

a. 36 Bytes FCB-Bereich
b. 28 Bytes Datei-Informationen
c. n Bytes Satz-Puffer-Bereich
d. m Bytes Schluesselbereich Indextabelle

Bei der Dimensionierung des Indextabellenbereiches ist zu beachten, dass insgesamt 64 Bytes plus die Laenge eines logischen Satzes fuer die eigentliche Indextabelle nicht nutzbar sind.

2) In die Indextabelle ist der Schluessel des letzten Satzes eines jeden Dateisegmentes eingetragen. Die Anzahl der Dateisegmente ergibt sich aus der maximalen Anzahl der Schluessel, die in die Indextabelle einspeicherbar sind. In KISAMC wird die Anzahl der Saetze im Segment (Segmentierungsfaktor) wie folgt berechnet:

$$SGM = \frac{\text{Anzahl\_logische\_Saetze\_der\_Datei} * \text{Schluessellaenge}}{\text{Laenge der Indextabelle}}$$

Das ganzzahlige Ergebnis dieser Berechnung wird um 1 erhoeht, wenn ein Rest bei der Division ensteht.
Die Anzahl der abzuspeichernden Schluessel, die gleich der Anzahl der Dateisegmente ADS ist, berechnit sich zu:

$$ADS = \frac{\text{Anzahl\_logische\_Saetze\_der\_Datei}}{SGM}$$

Das ganzzahlige Ergebnis wird bei einem Restwert ebenfalls um 1 erhoeht. Der Restwert besagt, dass es ein Restsegment gibt (letztes Segment), welches weniger Saetze enthaelt als der Segmentierungsfaktor SGM vorgibt.

2. Parameter fuer Aufruf

Parameter 1 (HL): Adresse der Zeichenkette, die die Angaben

- Funktionscode 01H
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

enthaelt.

Parameter 2 (DE): Adresse Indextabelle

Parameter 3 (BC): Adresse eines Feldes, auf dem 4 Zwei-Byte-Operanden (INTEGER-Werte) abgespeichert sind mit den Informationen:

- Schluessellaenge
- Schluesselposition
- logische Satzlaenge
- Laenge Indextabelle

3. Hinweise:

1. Die Rueckkehrcodes (im Byte 1 des ersten Parameters) sind nach Ablauf der Funktion entsprechend Abschnitt 5.4. auszuwerten.
2. Vor Aufruf der Funktion ist das Byte 1 des ersten Parameters mit dem Wert 01H zu belegen (Funktionscode OPEN setzen!).
3. Die Angabe der Schluesselposition bezieht sich immer auf die Anfangsadresse des gischen Satzes, die als Position 1 des logischen Satzes gewertet wird.

   Beispiel: Bei Schluesselposition 10 ist das zehnte Zeichen des logischen Satzes der Schluesselbeginn.

5.3.2. CLOSE-Funktion

1. Aufgabe

- Schreiben der Endekennung (FFFFH,1AH...1AH)
- Abschliessen der Datei

2. Parameter fuer Aufruf

Die CLOSE-Funktion verlangt nur einen Parameter (Register HL). Er ist die Anfangsadresse der fuer die Datei vereinbarten Zeichenkette mit dem Inhalt:

- Funktionscode 06H
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

3. Hinweise:

1. Die Rueckkehrcodes (im Byte 1 des ersten Parameters) sind nach Ablauf der Funktion entsprechend Abschnitt 5.4. auszuwerten.
2. Vor Aufruf der Funktion ist das Byte 1 des ersten Parameters mit dem Wert 06H zu belegen (Funktionscode CLOSE setzen!).

5.3.3. KEY-Funktion

1. Aufgabe

- Suchen eines logischen Satzes unter einem vorgegebenen Suchschluessel
- Uebergabe des gefundenen Datensatzes an das aufrufende Programm

Bemerkungen:

1) Der entsprechende Suchschluessel wird so lange mit den in der Indextabelle eingetragenen Schluesseln verglichen, bis ein Schluessel in der Indextabelle groesser oder gleich dem Suchschluessel ist.

2) Die Anzahl der ausgefuehrten Schluesselvergleiche -1 entspricht der Anzahl der Segmente, die vor dem entsprechenden Segment liegen, in dem sich der Satz befindet.

3) Um die Anzahl der vor dem zu durchsuchenden Segment liegenden logischen Saetze zu ermitteln, wird die Anzahl der Segmente, die vor dem zu durchsuchenden Segment liegen, mit dem Segmentierungsfaktor (Anzahl Saetze im Segment) multipliziert. Der sich ergebende Wert wird mit der logischen Satzlaenge multipliziert und durch die Sektorlaenge 128 dividiert. Das ist die echte physische Satzposition des ersten logischen Satzes des zu durchsuchenden Dateisegmentes.

4) Das ermittelte Dateisegment wird durchsucht bis entweder der Satz gefunden oder die Segmentgrenze erreicht ist. Im letzteren Falle erfolgt das sequentielle Suchen im Erweiterungsbereich der Datei (falls vorhanden). Wird auch dann der Satz nicht gefunden, erfolgt ueber den Rueckkehrcode die entsprechende Fehlermeldung.
Im Normalfall wird der gefundene Satz an das aufrufende Programm uebergeben.

2. Parameter fuer Aufruf

Parameter 1 (HL): Adresse der Zeichenkette, die die Angaben

- Funktionscode 02H
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

enthaelt.

Parameter 2 (DE): Adresse eines Vektors, der 2-Byte-Operanden (INTEGER-Feld) enthaelt und den zu suchenden Datensatz beschreibt (siehe Abschnitt 5.2.3.!)

Bemerkung:

Dieser Parameter uebergibt die Adressen der Satzelemente des zu suchenden Satzes im rufenden Programm, denn KEY vermittelt den gefundenen Satz an das rufende Programm und speichert die Satzelemente auf den vermittelten Adressen ab.

Parameter 3 (BC): Adresse des Suchschluessels

3. Hinweise

1. Die Rueckkehrcodes (im Byte 1 des ersten Parameters) sind nach Ablauf der Funktion entsprechend Abschnitt 5.4. auszuwerten.
2. Vor Aufruf der Funktion ist das Byte 1 des ersten Parameters mit dem Wert **02H** zu belegen (Funktionscode KEY setzen!).

5.3.4 READ-Funktion

1. Aufgabe

- Lesen eines logischen Satzes
- Uebergabe des Datensatzes an das aufrufende Programm

Bemerkungen:

1) Die Funktion READ liest einen logischen Satz ab der aktuellen Dateiposition. Unter der aktuellen Dateiposition ist der Zeiger auf den naechsten logischen Satz in der Datei nach einem erfolgten Zugriff zu verstehen. KISAMC vermerkt nach jedem Zugriff die aktuelle Dateiposition und stellt nach einem Zugriff auf die nachfolgende logische Satzposition ein. Somit ist durch einen zyklischen Aufruf der READ-Funktion nach dem Eroeffnen einer Datei das Vorwaertslesen der gesamten Datei moeglich. Die OPEN-Funktion stellt die aktuelle Dateiposition auf den ersten logischen Satz ein.
   Eine andere Arbeitsvariante ist das Vorwaertslesen ab der Dateipositon nach erfolgtem KEY-Zugriff.

2) Beim zyklischen Vorwaertslesen wird mit dem Rueckkehrcode **17H** die **EOF-Meldung** gegeben.

2. Parameter fuer Aufruf

Parameter 1 (HL): Adresse der Zeichenkette, die die Angaben

- Funktionscode **05H**
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

enthaelt.

Parameter 2 (DE): Adresse eines Vektors, der 2-Byte-Operanden (INTEGER-Feld) enthaelt und den zu lesenden Datensatz beschreibt (siehe Abschnitt 5.2.3.!)

Bemerkung:

Dieser Parameter uebergibt die Adressen der Satzelemente des zu lesenden Satzes im aufrufenden Programm. READ uebergibt den gelesenen Satz an das rufende Programm und speichert die Satzelemente auf den vermittelten Adressen ab.

3. Hinweise

1. Die Rueckkehrcodes (im Byte 1 des ersten Parameters) sind nach Ablauf der Funktion entsprechend Abschnitt 5.4. auszuwerten.
2. Vor Aufruf der Funktion ist das Byte 1 des ersten Parameters mit dem Wert 05H zu belegen (Funktionscode READ setzen!).

5.3.5. EXTEND-Funktion

1. Aufgabe

- Uebernahme eines logischen Satzes vom aufrufenden Programm
- Anfuegen des Satzes an die spezifizierte Datei

EXTEND bedeutet "Dateierweiterung" und realisiert das Anfuegen eines logischen Satzes an die Datei.

Bemerkungen:

1) KISAMC vermerkt sich beim Eroeffnen einer schluesselindizierten Datei die Dateiendeposition als Anfangsposition des Erweiterungsbereiches.

2) Die Dateiendeposition wird beim Anfuegen von Saetzen an die Datei unter KISAMC-Verwaltung erhoeht.

3) KISAMC testet nach dem Schreiben eines physischen Sektors (ueber BDOS-Funktion), ob die Datentraegergrenze erreicht ist. Tritt dieser Fall ein, wird die Datei automatisch auf dem aktuellen Stand abgeschlossen. Ein solcher Fall wird mit dem Rueckkehrcode 02H vermerkt.

4) Das Neuanlegen einer schluesselindizierten Datei erfolgt mit der EXTEND-Funktion:
In der OPEN-Funktion ist der Name der neu anzulegenden Datei und die Laufwerksbezeichnung anzugeben. Beim OPEN wird ueber die BDOS-Funktion 15 erkannt, dass diese Datei noch nicht existiert. KISAMC legt dann automatisch eine Datei unter dem Namen an und gibt den Rueckkehrcode 18H zurueck. Bei einem zyklischen Aufruf der EXTEND- Funktion wird die neue Datei geschrieben.
Es ist zu beachten, dass nach Neuerstellen einer Datei noch keine Indextabelle besteht. Die gesamte Datei gilt

dann als Erweiterung, wird folglich bei Aufruf der KEY-Funktion sequentiell durchsucht. Erst nach Schliessen der neuerstellten Datei und nachfolgendem Eroeffnen existiert die Indextabelle.

2. Parameter fuer Aufruf

Parameter 1 (HL): Adresse der Zeichenkette, die die Angaben

- Funktionscode 04H
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

enthaelt.

Parameter 2 (DE): Adresse eines Vektors, der 2-Byte-Operanden (INTEGER-Feld) enthaelt und den zu uebertragenden Datensatz beschreibt (siehe Abschnitt 5.2.3.1)

Bemerkung:

Dieser Parameter uebergibt die Adressen der Satzelemente des zu schreibenden Satzes vom aufrufenden Programm. KISAMC uebernimmt die Satzelemente des zu schreibenden Datensatzes und speichert diesen auf der Diskette ab.

3. Hinweise

1. Die Rueckkehrcodes (im Byte 1 des ersten Parameters) sind nach Ablauf der Funktion entsprechend Abschnitt 5.4. auszuwerten.
2. Vor Aufruf der Funktion ist das Byte 1 des ersten Parameters mit dem Wert 04H zu belegen (Funktionscode EXTEND setzen!).

5.3.6. UPDATE-Funktion

1. Aufgabe

- Rueckpositionieren auf den eben gelesenen Satz
- Uebernahme des neuen Satzinhaltes vom aufrufenden Programm
- Ueberschreiben des alten Satzes

UPDATE realisiert das Ueberschreiben eines ueber READ oder KEY gelesenen Satzes mit einem neuen Satzinhalt.

Bemerkungen:

Mit UPDATE kann eine Datei teilweise oder vollstaendig mit neuem Inhalt belegt werden. Weiter ist es moeglich, die aus der Datei zu loeschenden Saetze durch neue Satzinhalte zu

belegen, praktisch mit neuen Saetzen zu ueberschreiben, die sonst ueber Dateierweiterung an die Datei angefuegt werden muessten. Damit koennen fuer grosse Dateien sehr zeitaufwendige Regenerierungslaeufe erspart werden. Durch dieses Prinzip koennen alte Schluessel durch neue ueberschrieben werden. KISAMC besitzt aus diesem Grund auch keine Schluesselpruefung als Zugriffsschutz gegen Schluesselaenderungen.

2. Parameter fuer Aufruf

Parameter 1 (HL): Adresse der Zeichenkette, die die Angaben

- Funktionscode 03H
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

enthaelt.

Parameter 2 (DE): Adresse eines Vektors, der 2-Byte-Operanden (INTEGER-Feld) enthaelt und den zu uebertragenden Datensatz beschreibt (siehe Abschnitt 5.2.3.!)

Bemerkung:

Dieser Parameter uebergibt die Adressen der Satzelemente, des zu schreibenden Satzes vom aufrufenden Programm. KISAMC uebernimmt die Satzelemente des zu schreibenden Datensatzes und ueberschreibt den positionierten Datensatz.

3. Hinweise

1. Die Rueckkehrcodes (im Byte 1 des ersten Parameters) sind nach Ablauf der Funktion entsprechend Abschnitt 5.4. auszuwerten.

2. Vor Aufruf der Funktion ist das Byte 1 des ersten Parameters mit dem Wert 03H zu belegen (Funktionscode UPDATE setzen!).

5.4. Uebersicht ueber die Rueckkehrcodes

Gliederung: 1. Code
2. Bedeutung
3. erforderliche Reaktion im aufrufenden Programm

Rueckkehrcodes in der OPEN-Funktion

1. 00H
2. Fehlerfreie Abarbeitung der Funktion
3. Normale Weiterarbeit

1. 03H
2. Aufgrund eines Datei- oder Programmfehlers hat sich eine unzulaessige Dateiposition ergeben.
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
   - Datei untersuchen und Korrektur mit DIENST 1520 (SCPX) vornehmen.
   - Ueberpruefen der Datei-Parameter-Angaben im rufenden Programm

1. 0AH
2. Falscher Funktionscode im ersten Parameter vorgegeben (nur Werte von 01H...06H zulaessig)
3. - Pruefen der Funktionscodes vor jedem Funktionsaufruf
   - Kein CLOSE!

1. 0BH
2. Datei ist nicht sortiert
   Ursachen:
   - Falsche Schluesselvorgaben
   - Kein Sortierlauf vor Programmlauf
   - Binaerwert des Schluessels nicht zwischen 20H und 7FH
3. - Abbruch der Dateibearbeitung
   - Sortierung der Datei mittels SORT 1520 (SCPX)
   - Ueberpruefung der Schluessel und Schluesselparameter
   - Kein CLOSE!

1. 0DH
2. Dateiname enthaelt unzulaessige Zeichen
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
   - Aenderung der Zeichenkette, die den Dateinamen enthaelt

1. 0EH
2. Dateiname enthaelt keinen Punkt nach dem 8. Zeichen
3. Siehe 0DH!

1. 0FH
2. Anzahl der moeglichen Dateien ueberschritten (maximal 10)
3. - Abbruch der Dateibearbeitung
   - Ueberpruefen und Korrektur der OPEN-/CLOSE-Aufrufe im Anwenderprogramm

1. 88H
2. Lesefehler beim Lesen der Saetze zum Indextabellenaufbau
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
   - Pruefen und Korrektur der Datei mit DIENST 1520 (SCPX)

1. 10H
2. falscher Laufwerkscode im ersten Parameter vorgegeben (nur A, B, C, D, E moeglich; auch in Kleinbuchstaben)
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
   - Korrektur der Laufwerksangabe im rufenden Programm

1. 11H
2. Schluessellaenge groesser als die logische Satzlaenge
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
   - Korrektur der entsprechenden Datei-Spezifikationen

1. 12H
2. Schluesselposition liegt ausserhalb des logischen Satzes
3. siehe 11H !

1. 13H
2. Indextabelle zu klein (Bereich kleiner als eine Schluessellaenge)
3. siehe 11H !

1. 14H
2. Schluessellaenge groesser als 255 (FFH)
3. siehe 11H !

1. 15H
2. Anlegen einer Datei nicht moeglich
   Ursache:
   - Diskettenverzeichnis voll
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
   - Freie Datendiskette verwenden

1. 18H
2. Kein Fehlercode, sondern Mitteilung ueber das Anlegen einer neuen Datei
   Sie wird ausgegeben, wenn eine neue Datei angelegt werden soll (siehe EXTEND-Funktion). Diese Mitteilung erfolgt auch, wenn Laufwerkscode bzw. Diskette verwechselt wurden und auf der Diskette im angegebenen Laufwerk keine Datei unter dem Namen existiert.
3. Bei beabsichtigtem Neuanlegen einer Datei normale Weiterarbeit, sonst Abbruch der Dateibearbeitung

1. 22H
2. Datei unter dem gleichen Namen schon eroeffnet
   Ursache:
   KISAMC erlaubt keine gleichen Dateinamen!
3. siehe 0DH!

Rueckkehrcodes in der KEY-Funktion

1. 00H, analog OPEN-Funktion

1. 03H, analog OPEN-Funktion

1. 0AH, analog OPEN-Funktion

1. 0CH
2. Gesuchter Satz nicht in Datei
   Ursachen:
   - Falsche Parameterangaben
   - Falsche Dateigeneration verwendet
   - Kann auch beabsichtigt sein, um Datei zu pruefen
3. Behandlung entsprechend Ursache bzw. Erfordernis

1. 0DH, analog OPEN-Funktion

1. 0EH, analog OPEN-Funktion

1. 10H, analog OPEN-Funktion

1. 16H
2. Zugriff auf eine noch nicht eroeffnete oder bereits geschlossene Datei
   Ursachen:
   - Datei noch nicht eroeffnet
   - Dateiname bei OPEN anders angegeben als in Funktion
   - Durch Dateifehler wurde Datei automatisch geschlossen. Die Fehlercodeabfrage wurde bei der vorausgegangenen Funktion vergessen.
3. - Abbruch der Dateibearbeitung
   - Korrektur des Programmes entsprechend Ursache

1. 88H
2. Lesefehler bei KEY-Zugriff
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
     KISAMC hat die Datei automatisch geschlossen.
   - Pruefen und Korrektur der Datei mit DIENST 1520 (SCPX)

1. 19H
2. Diesem Fehler ist ein Lese-/Schreibfehler beim Ausfuehren der vorangegangenen Funktion vorausgegangen. Zusaetzlich trat beim automatischen Schliessen der Datei durch KISAMC beim Schreiben des letzten Sektors (Endekennung) ein Schreibfehler auf.
   - Die Datei konnte nicht geschlossen werden.
   - Die Datei ist eventuell nicht verwendbar!
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
   - Moegliches Retten der Datei mit DIENST 1520 (SCPX)

1. 21H
2. Widerspruch zwischen angegebener logischer Satzlaenge und der sich aus der Summe der Laengen aller Satzelemente ergebenden Laenge.
   Ursache:
   - Laengenangaben im Parameterfeld falsch
3. - CLOSE ausfuehren!
   - Abbruch der Dateibearbeitung
   - Korrektur der Parameterangaben

Rueckkehrcodes in der READ-Funktion

1. 00H, analog OPEN-Funktion

1. 03H, analog OPEN-Funktion

1. 0AH, analog OPEN-Funktion

1. 0DH, analog OPEN-Funktion

1. 0EH, analog OPEN-Funktion

1. 10H, analog OPEN-Funktion

1. 16H, analog KEY-Funktion

1. 17H
2. Dateiende bei Vorwaertslesen erreicht (EOF)!

1. 88H
2. Lesefehler
3. Analog KEY-Funktion

1. 19H, analog KEY-Funktion

1. 21H, analog KEY-Funktion

Rueckkehrcodes in der EXTEND-Funktion

1. 00H, analog OPEN-Funktion

1. 02H
2. Auf Diskette kein freier Platz verfuegbar
   Die Datei wurde automatisch von KISAMC abgeschlossen.
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!

1. 03H, analog OPEN-Funktion

1. 05H
2. Diskettenverzeichnis ist voll!
   Die Datei wurde automatisch von KISAMC abgeschlossen.
3. siehe 02H in EXTEND-Funktion!

1. 0AH, analog OPEN-Funktion

1. 0DH, analog OPEN-Funktion

1. 0EH, analog OPEN-Funktion

1. 10H, analog OPEN-Funktion

1. 16H, analog KEY-Funktion

1. 88H
2. Schreibfehler
3. Analog KEY-Funktion

1. 21H, analog KEY-Funktion

1. 19H, analog KEY-Funktion

Rueckkehrcodes_in_der_UPDATE-Funktion

1. 00H, analog OPEN-Funktion

1. 03H, analog OPEN-Funktion

1. 0AH, analog OPEN-Funktion

1. 0DH, analog OPEN-Funktion

1. 0EH, analog OPEN-Funktion

1. 10H, analog OPEN-Funktion

1. 16H, analog KEY-Funktion

1. 88H
2. Schreibfehler beim Rueckschreiben
3. Analog KEY-Funktion

1. 19H, analog KEY-Funktion

1. 21H, analog KEY-Funktion

Rueckkehrcodes_in_der_CLOSE-Funktion

1. 00H, analog OPEN- FUnktion

1. 03H, analog OPEN-Funktion

1. 0DH, analog OPEN-Funktion

1. 0EH, analog OPEN-Funktion

1. 10H, analog OPEN-Funktion

1. 0AH, analog OPEN-Funktion

1. 16H, analog KEY-Funktion

1. 19H,
2. Schreibfehler beim Rueckschreiben des letzten Sektors
3. - Abbruch der Dateibearbeitung
   - Kein CLOSE!
   - Pruefen und Korrektur der Datei mit DIENST 1520 (SCPX), falls verwendbar

1. 20H
2. Datei konnte bei Aufruf der CLOSE-Funktion nicht abgeschlossen werden
   Ursachen:
   - Falsches Laufwerk fuer Datei angegeben
   - Falsche Diskette
   - Diskette defekt

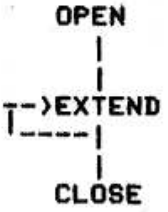
Hinweise:

1. Fuer die Orientierung ueber die Rueckkehrcodes im rufenden Programm ist der Nutzer selbst verantwortlich. Innerhalb der Nutzerprogramme sind fuer jede schluesselindizierte Datei entsprechende Mitteilungen und Abfragen zu programmieren, um ordnungsgemaesse Fehlerbehandlungen zu erreichen.
2. Alle aufgefuehrten Rueckkehrcodes beziehen sich auf die aktuelle Datei. Daraus ergibt sich, dass der Nutzer bei Auftreten von Fehlern fuer alle weiteren eroeffneten schluesselindizierten Dateien die erforderlichen CLOSE-Funktionen aufzurufen hat, um den aktuellen Stand zu retten.

5.5. Arbeitsvarianten mit KISAMC

Nachfolgende Beispiele sollen moegliche Anwendungen von KISAMC verdeutlichen.

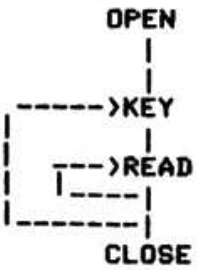
1.



2.

3.

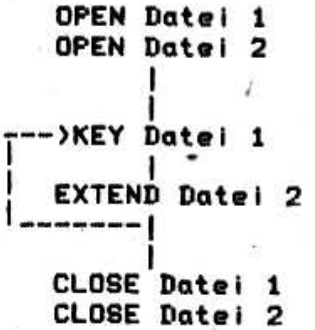
4.



5.

6.

7.



8.

zu 1.: - Neuerstellen einer Datei, falls die benannte Datei nicht existiert oder
- Erweitern Datei, wenn eine Datei unter dem Namen auf Diskette existiert

zu 2.: - sequentielles Lesen ab Dateianfang

zu 3.: - Suchen und Aendern eines Satzes
- Anschliessendes Rueckschreiben

zu 4.: - Suchen eines Satzes
- Lesen ab gefundenem Satz bis Dateiende

zu 5.: - Suchen bestimmter Saetze

zu 6.: - Lesen ab aktueller Dateiposition
- Rueckschreiben des gelesenen Satzes nach erfolgter Aenderung

zu 7.: - Suchen eines bestimmten Satzes in Datei 1
- Schreiben des gefundenen Satzes in Datei 2 (Anfuegen)

zu 8.: - Lesen eines Satzes ab aktueller Dateiposition aus Datei 1
- Schreiben des gelesenen Satzes in Datei 2 (Anfuegen)

5.6 Multivolume-Arbeit mit KISAMC

Allgemeine Hinweise

Die von KISAMC uebergebenen Rueckkehrcodes ermoeglichen eine effektive Mehrdatentraegerarbeit (Multivolumearbeit). Eine sich ueber mehrere Disketten erstreckende schluesselindizierte Datei ist als eine aus Teildateien bestehende Datei aufzufassen. Die Anzahl der Dateien bzw. Disketten ist bezueglich der Laufwerksanzahl begrenzt. Jede der Teildateien ist vor bzw. nach einem Dateibearbeitungsprozess mit SORT 1520 (SCPX) zu sortieren.
Es wird vorausgesetzt, dass die in einem Prozess zu bearbeitenden Teildateien mit unterschiedlichen Dateinamen benannt sind. Das SCPX gestattet keine Verarbeitung von Dateien mit gleichen Namen. Es empfielt sich, die Teildateien mit einem gleichen Stammnamen zu versehen, der um einen mit aufsteigender Nummer versehenen Zifernteil erweitert ist.

Beispiel:

```
DAT1.TST   - Teildatei 1
DAT2.TST   - Teildatei 2
DAT3.TST   - Teildatei 3
```

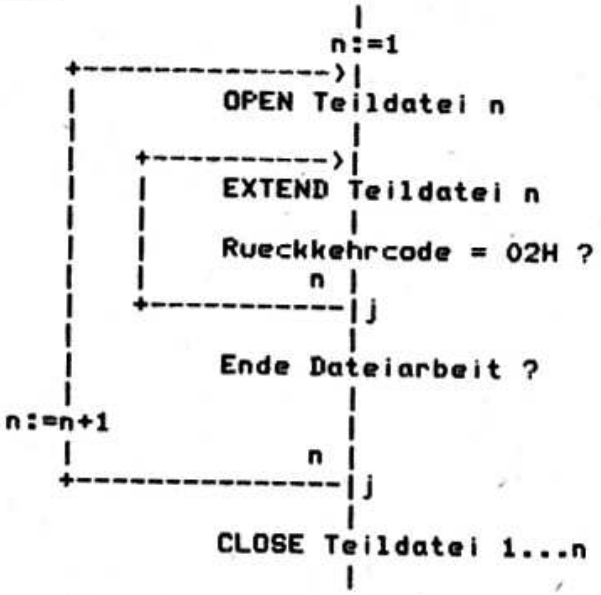
Fuer alle der in einem Prozess zu bearbeitenden Teildateien ist im Anwenderprogramm ein Bereich fuer die Indextabelle zu definieren.
Ausser fuer Erstellen bzw. Erweitern (Funktion EXTEND) einer Multivolumedatei sind alle Teildateien zu Beginn des Anwenderprogrammes nacheinander zu eroeffnen (OPEN-Funktion KISAMC). Eine neu anzulegende Multivolumedatei ist nur mit dem ersten und eine zu erweiternde mit dem letzten Datentraeger zu eroeffnen. Nach dem Eroeffnen folgt die im Anwenderprogramm vorgegebene Dateiarbeit mit KISAMC.
Nach Beenden der Dateiarbeitung mit KISAMC sind alle eroeffneten Teildateien mit der CLOSE-Funktion zu schliessen.
Der Diskettenwechsel ist waehrend der Dateibearbeitung moeglich. Vorauszusetzen ist, dass die abgearbeitete Teildatei geschlossen und die neu eingelegte Datei nach dem Diskettenwechsel durch Aufruf der OPEN-Funktion eroeffnet wird. Eine Multivolumearbeit mit laufendem Diskettenwechsel ist uneffektiv, da bei jedem OPEN die Indextabelle erneut aufgebaut werden muss.

5.6.1 Arbeitsweise beim Neuanlegen Multivolumedatei

Nach Eroeffnen der ersten Teildatei muss ein zyklisches EXTEND ueber KISAMC erfolgen. Bei Rueckmeldung des Rueckkehrcodes 02H (Datentraeger voll) muss ein OPEN fuer die auf dem nachfolgenden Datentraeger anzulegende Datei erfolgen.
Laufwerksbezeichnung und Dateibezeichnung sind bei Dateiwechsel vom Anwenderprogramm bereitzustellen.
Nach Beenden der Dateiarbeit sind alle erstellten Teildateien mit der CLOSE-Funktion zu schliessen.

Diskettenwechsel auf einem Laufwerk ist moeglich. Diese Arbeitsweise ist auf einen staendigen Dialog angewiesen. Vor Diskettenwechsel muss die Teildatei durch Aufruf der CLOSE-Funktion geschlossen werden.

Ablaufschema:



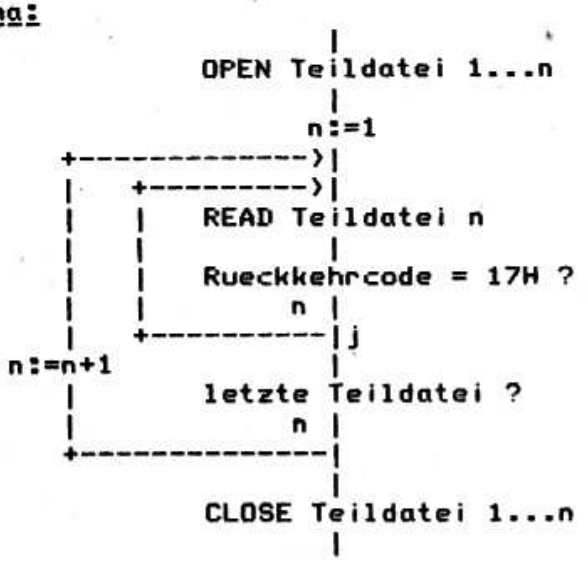
5.6.2 Arbeitsweise beim Erweitern Multivolumedatei

Es muessen nur dann alle Teildateien eroeffnet und verfuegbar sein, wenn ausser EXTEND noch andere Zugriffe auf die Multivolumedatei wie READ, KEY oder UPDATE erfolgen. Soll die Multivolumedatei nur erweitert werden, kann der letzte Datentraeger, auf dem noch freier Diskettenraum ist, verfuegbar gemacht und analog 5.6.1 gearbeitet werden (siehe Ablaufschema 5.6.1).

5.6.3 Arbeitsweise beim Lesen Multivolumedatei

Nachdem alle Teildateien in den Laufwerken eroeffnet wurden, ist ein mit der ersten Teildatei beginnender READ-Zyklus abzuarbeiten.

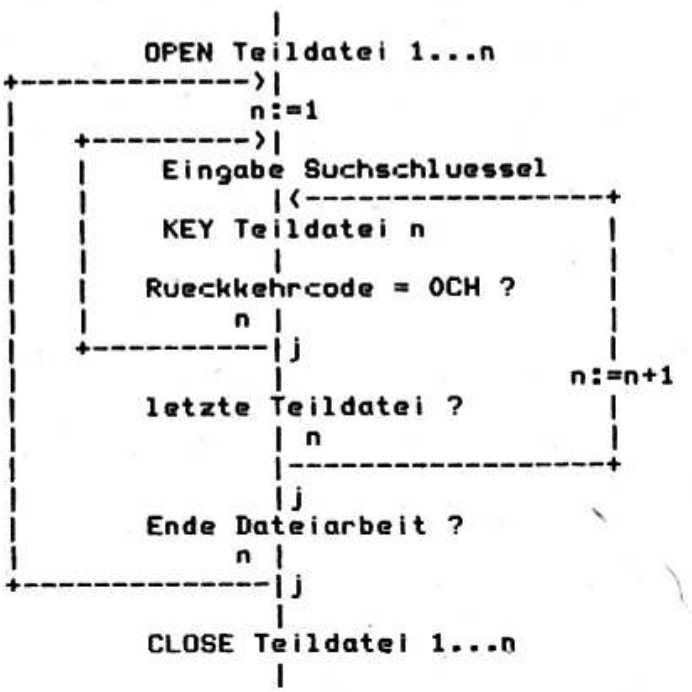
Ablaufschema:



5.6.4 Arbeitsweise mit schluesselindiziertem Zugriff auf eine Multivolumedatei

Nach Eroeffnen aller Teildateien werden diese mit Aufruf der KEY-Funktion nacheinander durchsucht. Bei nicht gefundenem Satz in einer Teildatei wird die naechste Teildatei durchsucht. Bringt auch die letzte Teildatei den Rueckkehrcode 0CH, so ist der Satz nicht in der Multivolumedatei enthalten.

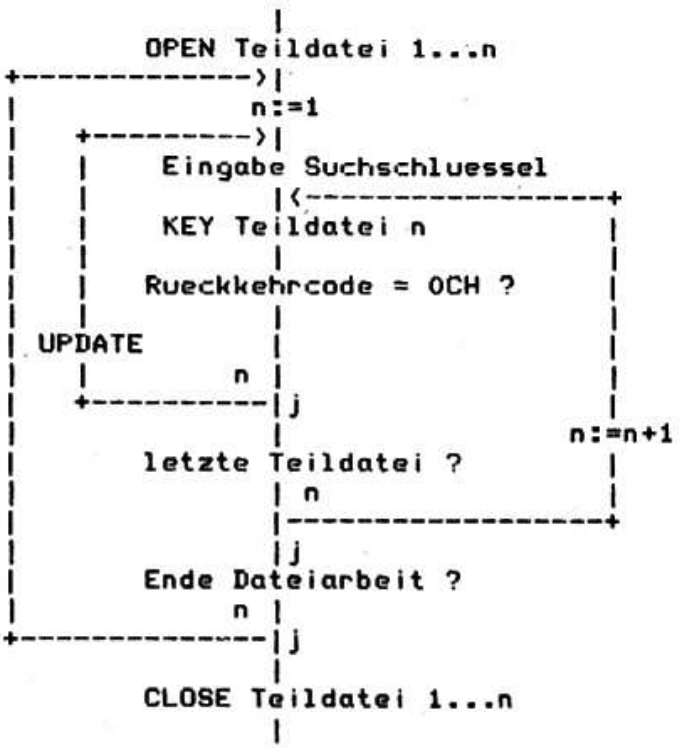
Ablaufschema:



5.6.5 Arbeitsweise mit UPDATE in einer Multivolumedatei

Da UPDATE sich immer auf einen zuvor ueber READ oder KEY gelesenen Satz bezieht, ergibt sich die Arbeitsweise entsprechend 5.6.3 bzw. 5.6.4. Der ueber READ oder KEY gelesene Satz wird mit der Funktion UPDATE ueberschrieben.

Ablaufschema:



Anhang - Anwendungsbeispiele KISAMC

1. Beispiele fuer aufrufende Assembler-Programme

1.1. OPEN-Funktion

OPEN verlangt:

Parameter 1 (HL) - Adresse der Zeichenkette fuer:

- Funktionscode
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

Parameter 2 (DE) - Adresse Indextabelle

Parameter 3 (BC) - Adresse eines INTEGER-Feldes mit den Informationen:

- Schluessellaenge
- Schluesselposition
- logische Satzlaenge
- Laenge Indextabelle

Realisierung:

1) Definition Zeichenkette (HL-Operand)

Beispiel:

```
ZK:   DB   00H                ;Funktionscode
      DB   "B:DAT.TST"        ;Laufwerk,Dateibezeich-
                              ;nung
```

2) Definition Indextabelle (DE-Operand)

Beispiel:

```
ITAB: DB  1000,00H
```

3) Definition des INTEGER-Feldes fuer Schluessellaenge, Schluesselposition, logische Satzlaenge, Laenge Indextabelle (BC-Operand)

Beispiel:

```
IVEKT:DW   12                 ;Schluessellaenge
      DW   10                 ;Schluesselposition
      DW   60                 ;logische Satzlaenge
      DW   1000               ;Laenge Indextabelle
```

4) Aufruf der OPEN-Funktion

Beispiel:

```
        LD    A,01H
        LD    (ZK),A          ;Setzen Funktionscode
        LD    HL,ZK           ;HL=Adresse Par.1
        LD    DE,ITAB         ;DE=Adresse Par.2
        LD    BC,IVEKT        ;BC=Adresse Par.3
        CALL  KISAMC          ;Aufruf OPEN-Funktion
        .
        .
```

1.2. KEY-Funktion

KEY verlangt:

Parameter 1 (HL) - Adresse der Zeichenkette fuer:

- Funktionscode
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

Parameter 2 (DE) - Adresse eines INTEGER-Feldes, auf dem die Datensatzinformationen

- Anzahl Datensatzelemente
- Adresse Satzelement 1
- Laenge Satzelement 1
- Adresse Satzelement 2
- Laenge Satzelement 2

  .
  .

- Adresse Satzelement n
- Laenge Satzelement n

fuer den zu suchenden Satz stehen.

Parameter 3 (BC) - Adresse Suchschluessel

Realisierung:

1) Definition Zeichenkette wie unter 1.1.

2) Definition Datensatzelemente und INTEGER-Feld zur Beschreibung des Datensatzes

Beispiel:

```
LOGS:  DS    60,00H           ;Datensatzbereich
ALOGS: DW    1                ;Anzahl Satzelemente=1
       DW    LOGS             ;Adresse Datensatz
       DW    60               ;Laenge Datensatz
```

Bemerkung: Hier liegt ein zusammenhaengender Datensatz vor, deshalb nur ein Datensatzelement!

3) Definition Suchschluessel

Beispiel:

```
SCHL: DB   "505-33-60000"
```

Bemerkung: Waehrend des Programmlaufes muss der Suchschluessel staendig aktualisiert werden!

4) Aufruf der KEY-Funktion

Beispiel:

```
LD   A,02H
LD   (ZK),A          ;Funktionscode KEY
LD   HL,ZK           ;HL=Adresse Par.1
LD   DE,ALOGS        ;DE=Adresse Par.2
LD   BC,SCHL         ;BC=Adresse Par.3
CALL KISAMC          ;Aufruf KEY-Funktion
.
.
```

1.3. READ-Funktion

READ verlangt:

Parameter 1 (HL) - Adresse der Zeichenkette fuer:

- Funktionscode
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

Parameter 2 (DE) - Adresse eines INTEGER-Feldes, auf dem die Datensatzinformationen fuer den zu lesenden Satz stehen (siehe KEY!).

Realisierung:

1) Definition Zeichenkette (HL-Operand)

Beispiel:

```
ZK:   DB   00H              ;Funktionscode
      DB   "C:ISDAT.LES"    ;Laufwerk,Dateibezeich-
                            ;nung
```

2) Definition Datensatzelemente und INTEGER-Feld zur Beschreibung des Datensatzes

Beispiel:

```
SATZ: DS   185,00H          ;Datensatzbereich
```

```
      ASZ: DW   1                ;Anzahl Satzelemente=1
           DW   SATZ             ;Adresse Datensatz
           DW   185              ;Laenge Datensatz
```

3) Aufruf der READ-Funktion

Beispiel:

```
           LD   A,05H
           LD   (ZK),A           ;Funktionscode READ
           LD   HL,ZK            ;HL=Adresse Par.1
           LD   DE,ASZ           ;DE=Adresse Par.2
           CALL KISAMC           ;Aufruf READ-Funktion
           .
           .
```

1.4. EXTEND-Funktion

EXTEND verlangt:

Parameter 1 (HL) - Adresse der Zeichenkette fuer:

- Funktionscode
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

Parameter 2 (DE) - Adresse eines INTEGER-Feldes, auf dem die Datensatzinformationen fuer den zu schreibenden Satz stehen (siehe KEY!).

Realisierung:

1) Definition Zeichenkette (HL-Operand)

Beispiel:

```
      ZK:  DB   00H              ;Funktionscode
           DB   "D:EXDAT.TST"    ;Laufwerk,Dateibezeich-
                                 ;nung
```

2) Definition Datensatzelemente und INTEGER-Feld zur Beschreibung des Datensatzes

Beispiel:

```
     SE1: DS   8
     SE2: DB   "505-33-99/6000"  ;Satzelement 2
     SE3: DS   40                ;Satzelement 3

      AS: DW   3                 ;Anzahl Satzelemente=3
          DW   SE1               ;Adresse Satzelement 1
          DW   8                 ;Laenge Satzelement 1
          DW   SE2               ;Adresse Satzelement 2
          DW   14                ;Laenge Satzelement 2
          DW   SE3               ;Adresse Satzelement 3
          DW   40                ;Laenge Satzelement 3
```

3) Aufruf der EXTEND-Funktion

Beispiel:

```
        LD    A,04H
        LD    (ZK),A          ;Funktionscode EXTEND
        LD    HL,ZK           ;HL=Adresse Par.1
        LD    DE,AS           ;DE=Adresse Par.2
        CALL  KISAMC          ;Aufruf EXTEND-Funktion
        .
        .
```

1.5. UPDATE-Funktion

UPDATE verlangt:

Parameter 1 (HL) - Adresse der Zeichenkette fuer:

- Funktionscode
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

Parameter 2 (DE) - Adresse eines INTEGER-Feldes, auf dem die Datensatzinformationen fuer den zu schreibenden Satz stehen (siehe KEY!).

Realisierung:

1) Definition Zeichenkette (HL-Operand)

Beispiel:

```
     ZK:   DB    00H             ;Funktionscode
           DB    "A:TEST.UPD"    ;Laufwerk,Dateibezeich-
                                 ;nung
```

2) Definition Datensatzelemente und INTEGER-Feld zur Beschreibung des Datensatzes

Beispiel:

```
        EL1: DB   10              ;Satzelement 1
        EL2: DB   8               ;Satzelement 2
        EL3: DB   8               ;Satzelement 3
        EL4: DB   12              ;Satzelement 4

        FELD:DW   4               ;Anzahl Satzelemente=4
             DW   EL1             ;Adresse Satzelement 1
             DW   10              ;Laenge Satzelement 1
             DW   EL2             ;Adresse Satzelement 2
             DW   8               ;Laenge Satzelement 2
             DW   EL3             ;Adresse Satzelement 3
             DW   8               ;Laenge Satzelement 3
             DW   EL4             ;Adresse Satzelement 4
             DW   12              ;Laenge Satzelement 4
```

3) Aufruf der UPDATE-Funktion

Beispiel:

```
        LD    A,03H
        LD    (ZK),A          ;Funktionscode UPDATE
        LD    HL,ZK           ;HL=Adresse Par.1
        LD    DE,FELD         ;DE=Adresse Par.2
        CALL  KISAMC          ;Aufruf UPDATE-Funktion
        .
        .
```

1.6. CLOSE-Funktion

CLOSE verlangt nur einen Parameter:

Parameter (HL) - Adresse der Zeichenkette fuer:

- Funktionscode
- Laufwerk einschliesslich Doppelpunkt
- Dateibezeichnung

Realisierung:

1) Definition Zeichenkette (HL-Operand)

Beispiel:

```
ZK:     DB    00H             ;Funktionscode
        DB    "A:TEST.DAT"    ;Laufwerk,Dateibezeich-
                              ;nung
```

2) Aufruf der CLOSE-Funktion

Beispiel:

```
        LD    A,06H
        LD    (ZK),A          ;Funktionscode CLOSE
        LD    HL,ZK           ;HL=Adresse ZK
        CALL  KISAMC          ;Aufruf CLOSE-Funktion
        .
        .
```

Hinweis fuer Assembler-Programmierer:

Der vom Betriebssystem SCPX standardmaessig bereitgestellte STACK ist bei der Nutzung von KISAMC nicht ausreichend. Deshalb ist die Definition eines eigenen STACK's im Nutzerprogramm unerlaesslich (siehe nachfolgende Beispielprogramme!).

```
        ;*** TEST-PROGRAMM FUER KISAMC ***
        ;***      EXTEND-Funktion       ***

;Demonstration der EXTEND-Funktion

;Inhalt:
;Die Datei "ISDATEI1.TST" wird auf der im Laufwerk B befindli-
;chen Diskette neu angelegt.  Es werden im Zyklus insgesamt 48
;Saetze der Laenge 60 aufgezeichnet. Der Schluessel wird durch
;ein  kleines Generierungsprogramm staendig erhoeht,  um jedem
;Datensatz seinen speziellen Schluessel zuzuordnen.

        extrn   kisamc
        ld      hl,0                ;Retten SP und eigenen
        add     hl,sp               ;Stack anlegen
        ld      (stkptr),hl
        ld      sp,stkptr-2

        ld      de,amld
        ld      c,9
        call    5
        ld      iy,0
        ld      (zwsp),iy           ;Zaehler Schreibzyklen
        ld      hl,dnlw             ;Parameter OPEN
        ld      de,itab
        ld      bc,schll
        ld      a,01h               ;Funktionscode OPEN
        ld      (dnlw),a
        call    kisamc              ;OPEN-Funktion
        ld      a,(dnlw)
        cp      18h                 ;Rueckkehrcodes testen
        jp      z,cslup
        cp      00h
        jp      nz,cfeco
cslup:  call    slup                ;Schluesselgenerierung
        ld      a,04h               ;Funktionscode EXTEND
        ld      (dnlw),a
        ld      hl,dnlw             ;Parameter EXTEND
        ld      de,alogs
        call    kisamc              ;EXTEND-Funktion
        ld      a,(dnlw)
        cp      00h                 ;Rueckkehrcode testen
        jp      z,nsatz
        cp      88h
        jp      z,cfeco
        cp      06h
        jp      m,cfeco
        ld      c,9
        call    feco                ;Fehlercode anzeigen
        ld      de,kette
        call    5
        ld      c,9
        ld      de,konst
        call    5
        jp      close
```

```
nsatz:  ld      iy,(zwsp)       ;Zaehler geschriebene
        inc     iy              ;Saetze
        ld      (zwsp),iy
        push    iy
        pop     hl
        ld      bc,30h
        xor     a
        sbc     hl,bc
        jp      nz,cslup        ;Endebedingung testen

close:  ld      a,06h           ;Funktionscode CLOSE
        ld      (dnlw),a
        ld      hl,dnlw         ;Parameter CLOSE
        call    kisamc          ;CLOSE-Funktion

cfeco:  call    feco            ;Anzeige Rueckkehrcode
        ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,konst
        call    5
        ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,emld         ;Endemeldung
        call    5

        ld      sp,(stkptr)     ;STACK-Pointer ruecksetzen

        jp      0

        ;Definitionen

kette:  db      0ah,0dh,24h
dnlw:   db      00h
        db      "b:isdatei1.tst"
emld:   db      "ENDE PROGRAMM"
schll:  dw      12
spos:   dw      10
lsl:    dw      60
litab:  dw      4096
itab:   ds      4*1024,00h
schls:  db      "101253000000"
logs:   ds      9,41h
        ds      12,00h
        ds      39,42h
amld:   db      "TEST KISAMC  *** EXTEND-Funktion ****"
zwsp:   ds      2
alogs:  dw      1
        dw      logs
        dw      60
konst:  db      "RUECKKEHRCODE:
rcb:    ds      2,00h
        db      24h
```

```
stack:  ds      50,00h          ;Stack
stkptr: dw      0               ;Rettebereich System-Stack

        ;UP fuer Generierung Schluessel

slup:   push    hl
        push    de
        push    bc
        ld      hl,schls+11
nst:    xor     a
        ld      a,m
        and     0fh
        inc     a
        cp      0ah
        jp      c,klz
        ld      a,30h
        ld      m,a
        dec     hl
        jp      nst
klz:    or      30h
        ld      m,a
        ld      de,logs+9
        ld      hl,schls
        ld      bc,(schll)
        ldir
        pop     bc
        pop     de
        pop     hl
        ret

        ;UP FECO - Ausgabe Fehlercode

feco:   ld      a,(dnlw)
        ld      hl,sutb
suzy:   cp      m
        jp      z,gefz
        inc     hl
        inc     hl
        inc     hl
        jp      suzy
gefz:   inc     hl
        ld      a,m
        ld      (rcb),a
        inc     hl
        ld      a,m
        ld      (rcb+1),a
        ret

sutb:   db      00h,30h,30h
        db      02h,30h,32h
        db      03h,30h,33h
        db      05h,30h,35h
        db      88h,38h,38h
        db      0ah,30h,41h
        db      0bh,30h,42h
```

```
db      0ch,30h,43h
db      0dh,30h,44h
db      0eh,30h,45h
db      0fh,30h,46h
db      10h,31h,30h
db      11h,31h,31h
db      12h,31h,32h
db      13h,31h,33h
db      14h,31h,34h
db      15h,31h,35h
db      16h,31h,36h
db      17h,31h,37h
db      18h,31h,38h
db      19h,31h,39h
db      20h,32h,30h
db      21h,32h,31h
db      22h,32h,32h
end
```

```
        ;*** TEST-PROGRAMM FUER KISAMC ***
        ;***      READ-Funktion         ***

;Demonstration der READ-Funktion

;Inhalt:
;Die Datei "ISDATEI1.TST" wird von der im Laufwerk B befindli-
;chen  Diskette  ab Dateianfang bis Dateiende  gelesen.  Jeder
;Satz  wird auf dem Bildschirm angezeigt.  Das Programm  endet
;mit der EOF-Meldung, Rueckkehrcode 17H.


        extrn   kisamc
        ld      hl,0            ;Retten SP und eigenen
        add     hl,sp           ;Stack anlegen
        ld      (stkptr),hl
        ld      sp,stkptr-2

        ld      de,amld
        ld      c,9
        call    5
        ld      iy,0
        ld      (zwsp),iy       ;Zaehler Lesezyklen
        ld      hl,dnlw         ;Parameter OPEN
        ld      de,itab
        ld      bc,schl1
        ld      a,01h           ;Funktionscode OPEN
        ld      (dnlw),a
        call    kisamc          ;OPEN-Funktion
        ld      a,(dnlw)
        cp      0               ;Rueckkehrcode testen
        jp      nz,aus1
cread:  ld      a,05h           ;Funktionscode READ
        ld      (dnlw),a
        ld      hl,dnlw         ;Paramter READ
        ld      de,alogs
        call    kisamc          ;READ-Funktion
        ld      a,(dnlw)
        cp      0               ;Rueckkehrcode testen
        jp      z,nsatz
        cp      88h
        jp      z,aus1
        cp      06h
        jp      m,aus1
        call    feco            ;Fehlercode anzeigen
        ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,konst
        call    5
        jp      close
nsatz:  ld      c,9
        ld      de,kette
        call    5
        ld      c,9
```

```
        ld      de,logs         ;Anzeige gelesenen Satz
        call    5
        ld      iy,(zwsp)
        inc     iy
        ld      (zwsp),iy       ;Zaehler Endebedingung
        push    iy
        pop     hl
        ld      bc,30h
        xor     a
        sbc     hl,bc
        jp      nz,cread        ;Endebedingung testen
close:  ld      a,06h           ;Funktionscode CLOSE
        ld      (dnlw),a
        ld      hl,dnlw         ;Parameter CLOSE
        call    kisamc          ;CLOSE-Funktion
ausl:   call    feco            ;Rueckkehrcode anzeigen
        ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,konst
        call    5

        ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,emld         ;Endemeldung
        call    5

        ld      sp,(stkptr)     ;Ruecksetzen STACK-Pointer

        jp      0

        ;Definitionen

kette:  db      0dh,0ah,24h
konst:  db      "RUECKKEHRCODE: "
rcb:    db      00h,00h,24h
dnlw:   db      00h
        db      "b:isdatei1.tst",0
emld:   db      "ENDE PROGRAMM",0
schll:  dw      12
spos:   dw      10
lsl:    dw      60
litab:  dw      4096
itab:   ds      4*1024,00h
schls:  db      "101253000000"
logs:   ds      60,00h
        db      24h
amld:   db      "TEST KISAMC  *** READ-Funktion ****"
zwsp:   ds      2
alogs:  dw      1
        dw      logs
        dw      60
stack:  ds      50,00h          ;Stack
```

```
stkptr: dw      0               ;Kettebereich System-Stack

        ;UP FECO - Ausgabe Fehlercode

feco:   ld      a,(dnlw)
        .
        .
        Siehe Testprogramm EXTEND Funktion!
        .
        .
        ret

sutb:   db      00h,30h,30h
        db      02h,30h,32h
        .
        .
        Siehe Testprogramm EXTEND-Funktion!
        .
        .
        end
```

```
          ;*** TEST-PROGRAMM FUER KISAMC ***
          ;***        KEY-Funktion        ***

;Demonstration der KEY-Funktion

;Inhalt:
;Aus der Datei "ISDATEI1.TST" (Laufwerk B) wird ueber die
;Funktion KEY jeder dritte Satz gelesen. Das Unterprogramm
;"SLUP" generiert fuer jeden dritten Satz den Suchschluessel.
;Jeder gefundene Satz wird ueber Bildschirm angezeigt. Das
;Programm endet mit dem Rueckkehrcode 0CH, weil im letzten
;Satz auf einen nicht in der Datei enthaltenen Schluessel
;zugegriffen werden soll.

          extrn   kisamc
          ld      hl,0
          add     hl,sp             ;Retten SP und eigenen
          ld      (stkptr),hl       ;Stack anlegen
          ld      sp,stkptr-2

          ld      de,amld
          ld      c,9
          call    5
          ld      iy,0
          ld      (zwsp),iy         ;Zaehler Suchzyklen
          ld      hl,dnlw           ;Parameter OPEN
          ld      de,itab
          ld      bc,schl1
          ld      a,01h
          ld      (dnlw),a          ;Funktionscode OPEN
          call    kisamc
          ld      a,(dnlw)          ;OPEN-Funktion
          cp      0
          jp      nz,aus1           ;Rueckkehrcode testen

cslup:    call    slup              ;Schluesselgenerierung
          ld      a,02h             ;Funktionscode KEY
          ld      (dnlw),a
          ld      hl,dnlw           ;Parameter KEY
          ld      bc,schls
          ld      de,alogs
          call    kisamc
          ld      a,(dnlw)          ;KEY-Funktion
          cp      0
          jp      z,nsatz           ;Rueckkehrcode testen
          cp      88h
          jp      z,aus1
          cp      06h
          jp      m,aus1
          call    feco              ;Fehlercode anzeigen
          ld      c,9
          ld      de,kette
          call    5
          ld      c,9
          ld      de,konst
          call    5
```

```
        jp      close

nsatz:  ld      c,9
        ld      de,kette
        call    5
        ld      c,9             ;Ausgabe des gefundenen Satzes
        ld      de,logs
        call    5

        ld      iy,(zwsp)
        inc     iy
        inc     iy              ;Inkrementieren Zaehler
        inc     iy              ;Programmzyklen
        ld      (zwsp),iy
        push    iy
        pop     hl
        ld      bc,33h
        xor     a
        sbc     hl,bc
        jp      nz,cslup        ;Endebedingung testen

close:  ld      a,06h           ;Funktionscode CLOSE
        ld      (dnlw),a
        ld      hl,dnlw         ;Parameter CLOSE
        call    kisamc          ;CLOSE-Funktion
ausl:   call    feco            ;Anzeige Rueckkehrcode
        ld      c,9
        ld      de,kette
        call    5               ;Kursorpositionierung

        ld      c,9
        ld      de,konst
        call    5

        ld      c,9
        ld      de,kette
        call    5

        ld      c,9
        ld      de,emld         ;Endemeldung
        call    5

        ld      sp,(stkptr)     ;STACK-Pointer ruecksetzen

        jp      0

        ;Definitionen

kette:  db      0ah,0dh,24h
dnlw:   db      00h
        db      "b:isdatei1.tstx"
emld:   db      "ENDE PROGRAMMx"
schll:  dw      12
spos:   dw      10
lsl:    dw      60
litab:  dw      4096
```

```
itab:   ds      4*1024,00h
schls:  db      "101253000000"
logs:   ds      60,00h
        db      "x"
amld:   db      "TEST KISAMC  *** KEY-Funktion ****"
zwsp:   ds      2
zw2:    ds      2
alogs:  dw      1
        dw      logs
        dw      60
konst:  db      "RUECKKEHRCODE: "
rcb:    db      00h,00h
        db      24h
stack:  ds      50,00h          ;Stack
stkptr: dw      0               ;Rettebereich System-Stack

        ;UP fuer Generierung Schluessel

slup:   push    hl
        push    de
        push    bc
        ld      a,03h
        ld      (zw2),a
ldhlb:  ld      hl,schls+11
nst:    xor     a
        ld      a,m
        and     0fh
        inc     a
        cp      0ah
        jp      c,klz
        ld      a,30h
        ld      m,a
        dec     hl
        jp      nst
klz:    or      30h
        ld      m,a
        ld      de,logs+9
        ld      hl,schls
        ld      bc,(schll)
        ldir
        xor     a
        ld      a,(zw2)
        dec     a
        ld      (zw2),a
        cp      00h
        jp      nz,ldhlb
        pop     bc
        pop     de
        pop     hl
        ret
```

```
        ;UP FECO - Ausgabe Fehlercode

feco:   ld      a,(dnlw)
        .
        .
        Siehe Testprogramm EXTEND-Funktion!
        .
        .
        ret

sutb:   db      00h,30h,30h
        db      02h,30h,32h
        .
        .
        Siehe Testprogramm EXTEND-Funktion!
        .
        .
        end
```

```
        ;*** TEST-PROGRAMM FUER KISAMC ***
        ;***      UPDATE-Funktion       ***

;Demonstration der UPDATE-Funktion

;Inhalt:
;Jeder dritte Satz der Datei "ISDATEI1.TST" wird von der im
;Laufwerk B befindlichen Diskette durch KEY gelesen. Die
;gesuchten Saetze werden geaendert und ueber die Funktion
;UPDATE zurueckgeschrieben. Gesuchter und geaenderter Satz
;werden jeweils zusammen ueber Bildschirm ausgegeben. Die
;Schluesselgenerierung fuer jeden dritten Satz erfolgt im
;Unterprogramm "SLUP".

        extrn   kisamc
        ld      hl,0            ;Retten SP und eigenen
        add     hl,sp           ;Stack anlegen
        ld      (stkptr),hl
        ld      sp,stkptr-2

        ld      de,amld
        ld      c,9
        call    5
        ld      iy,0
        ld      (zwsp),iy       ;Zaehler UPDATE-Laeufe
        ld      hl,dnlw         ;Parameter OPEN
        ld      de,itab
        ld      bc,schll
        ld      a,01h           ;Funktionscode OPEN
        ld      (dnlw),a
        call    kisamc          ;OPEN-Funktion
        ld      a,(dnlw)
        cp      0               ;Rueckkehrcode testen
        jp      nz,aus1

cslup:  call    slup            ;Schluesselgenerierung
        ld      a,02h           ;Funktionscode KEY
        ld      (dnlw),a
        ld      hl,dnlw         ;Parameter KEY
        ld      bc,schls
        ld      de,alogs
        call    kisamc          ;KEY-Funktion
        ld      a,(dnlw)
        cp      0               ;Rueckkehrcode abfragen
        jp      z,upd
        cp      88h
        jp      z,aus1
        cp      06h
        jp      m,aus1
        call    feco            ;Fehlercode anzeigen
        ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,konst
        call    5
```

```
        jp      close
upd:    ld      c,9
        ld      de,kette
        call    5
        ld      c,9             ;Ausgabe des gefundenen Satzes
        ld      de,logs
        call    5

        ld      c,9
        ld      de,mkey
        call    5

        ld      hl,schls        ;Uebertragen Schluessel in
        ld      bc,(schll)      ;rueckzuschreibenden Satz
        ld      de,logs2+9
        ldir

        ld      a,03h           ;Funktionscode UPDATE
        ld      (dnlw),a
        ld      hl,dnlw         ;Parameter UPDATE
        ld      de,alog2
        call    kisamc          ;Funktion UPDATE
        ld      a,(dnlw)
        cp      0               ;Rueckkehrcode testen
        jp      z,nupd

        cp      88h
        jp      z,aus1
        cp      06h
        jp      m,aus1
        call    feco            ;Fehlercode anzeigen
        ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,konst
        call    5
        jp      close

nupd:   ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,logs2        ;Anzeige des rueckgeschriebe-
        call    5               ;nen Satzes
        ld      c,9
        ld      de,mupd
        call    5
        ld      iy,(zwsp)
        inc     iy
        inc     iy
        inc     iy
        ld      (zwsp),iy       ;Zaehler Endebedingung
        push    iy
        pop     hl
        ld      bc,30h
```

```
        xor     a
        sbc     hl,bc
        jp      nz,cslup        ;Endebedingung testen

close:  ld      a,06h           ;Funktionscode CLOSE
        ld      (dnlw),a
        ld      hl,dnlw         ;Parameter CLOSE
        call    kisamc          ;CLOSE-Funktion

ausl:   call    feco            ;Anzeige Rueckkehrcode
        ld      c,9
        ld      de,kette
        call    5               ;Kursorpositionierung

        ld      c,9
        ld      de,konst
        call    5

        ld      c,9
        ld      de,kette
        call    5

        ld      c,9
        ld      de,emld         ;Endemeldung
        call    5

        ld      sp,(stkptr)     ;STACK-Pointer ruecksetzen

        jp      0

        ;Definitionen

kette:  db      0ah,0dh,24h
dnlw:   db      00h
        db      "b:isdatei1.tst*"
emld:   db      "ENDE PROGRAMM*"
schll:  dw      12
spos:   dw      10
lsl:    dw      60
litab:  dw      4096
itab:   ds      4*1024,00h
schls:  db      "101253000000"
logs:   ds      60,00h
        db      "*"
amld:   db      "TEST KISAMC  *** UPDATE-Funktion ****"
zwsp:   ds      2
zw2:    ds      2
alogs:  dw      1
        dw      logs
        dw      60
alog2:  dw      1
        dw      logs2
        dw      60
konst:  db      "RUECKKEHRCODE: "
rcb:    db      00h,00h,24h
stack:  ds      50,00h          ;Stack
```

```
stkptr: dw      0               ;Rettebereich System-Stack
mkey:   db      "  KEY*"
mupd:   db      "  UPDATE*"
logs2:  ds      9,43h
        ds      12,00h
        ds      39,44h
        db      24h

        ;UP fuer Generierung Schluessel

slup:   push    hl
        .
        .
        Siehe Testprogramm KEY-Funktion!
        .
        .
        ret

        ;UP FECO - Ausgabe Fehlercode

feco:   ld      a,(dnlw)
        .
        .
        Siehe Testprogramm EXTEND-Funktion!
        .
        .
        ret

sutb:   db      00h,30h,30h
        db      02h,30h,32h
        .
        .
        Siehe Testprogramm EXTEND-Funktion!
        .
        .
        end
```

```
        ;***      TEST-PROGRAMM FUER KISAMC        ***
        ;*** ARBEIT MIT ZWEI DATEIEN (KEY/EXTEND) ***

;Demonstration der Parallelarbeit

;Inhalt:
;Jeder dritte Satz der Datei "ISDATEI1.TST" (Laufwerk B)
;wird ueber die Funktion KEY durch Generierung des entspre-
;chenden Schluessels gelesen und in die Datei "DAT.TST" (Lauf-
;werk B) geschrieben. Alle,durch KEY gelesenen Saetze werden
;auf dem Bildschirm angezeigt.

        extrn   kisamc
        ld      hl,0                ;Retten SP und eigenen
        add     hl,sp               ;Stack anlegen
        ld      (stkptr),hl
        ld      sp,stkptr-2

        ld      de,amld
        ld      c,9
        call    5

        ld      iy,0
        ld      (zwsp),iy           ;Zaehler Durchlaeufe

        ld      hl,dnlw             ;Parameter OPEN Datei 1
        ld      de,itab
        ld      bc,schl1
        ld      a,01h               ;Funktionscode OPEN
        ld      (dnlw),a
        call    kisamc              ;OPEN Datei 1

        ld      a,(dnlw)
        cp      0                   ;Rueckkehrcode testen
        jp      nz,aus1

        ld      hl,dnlw2            ;Parameter OPEN Datei 2
        ld      de,itab2
        ld      bc,schl2
        ld      a,01h               ;Funktionscode OPEN
        ld      (dnlw2),a
        call    kisamc              ;OPEN Datei 2

        ld      a,(dnlw2)
        cp      18h                 ;Rueckkehrcode testen
        jp      z,nzyk
        cp      00h
        jp      nz,aus1

nzyk:   call    slup                ;Schluesselgenerierung
        ld      a,02h               ;Funktionscode KEY
        ld      (dnlw),a
        ld      hl,dnlw             ;Parameter KEY
        ld      bc,schls
        ld      de,alogs
```

```
        call    kisamc          ;KEY-Funktion

        ld      a,(dnlw)
        cp      0               ;Rueckkehrcode testen
        jp      z,asatz
        cp      88h
        jp      z,aus1
        cp      06h
        jp      m,aus1
        call    feco            ;Fehlercode Datei 1.anzeigen
        ld      c,9
        ld      de,kette
        call    5
        ld      c,9
        ld      de,konst
        call    5
        jp      close

asatz:  ld      c,9
        ld      de,kette
        call    5

        ld      c,9             ;Ausgabe des gefundenen Satzes
        ld      de,logs
        call    5

        ld      a,04h           ;Funktionscode EXTEND Datei 2
        ld      (dnlw2),a
        ld      hl,dnlw2        ;Parameter EXTEND Datei 2
        ld      de,alogs
        call    kisamc          ;EXTEND Datei 2

        ld      a,(dnlw2)
        cp      00h             ;Rueckkehrcode testen
        jp      z,eabfr
        cp      88h
        jp      z,aus1
        cp      06h
        jp      m,aus1
        ld      c,9
        call    feco            ;Fehlercode Datei 2 anzeigen
        ld      de,kette
        call    5
        ld      c,9
        ld      de,konst
        call    5
        jp      close

eabfr:  ld      iy,(zwsp)       ;Inkrementieren Zaehler
        inc     iy              ;fuer Durchlaeufe
        inc     iy
        inc     iy
        ld      (zwsp),iy
        push    iy
        pop     hl
        ld      bc,30h
```

```
        xor     a
        sbc     hl,bc
        jp      nz,nzyk         ;Endebedingung testen

close:  ld      a,06h           ;Funktionscode CLOSE
        ld      (dnlw),a
        ld      hl,dnlw         ;Parameter CLOSE
        call    kisamc          ;CLOSE Datei 1

        ld      a,06h           ;Funktionscode CLOSE
        ld      (dnlw2),a
        ld      hl,dnlw2        ;Parameter CLOSE
        call    kisamc          ;CLOSE Datei 2

aus1:   call    feco            ;Anzeige Rueckkehrcode
        ld      c,9
        ld      de,kette
        call    5               ;Kursorpositionierung

        ld      c,9
        ld      de,konst
        call    5

        ld      c,9
        ld      de,kette
        call    5

        ld      c,9
        ld      de,emld         ;Endemeldung
        call    5

        ld      sp,(stkptr)     ;STACK-Pointer ruecksetzen

        jp      0


        ;Definitionen

kette:  db      0ah,0dh,24h
dnlw:   db      00h
        db      "b:isdatei1.tst"
emld:   db      "ENDE PROGRAMM"
dnlw2:  db      00h
        db      "b:dat.tst"
itab2:  ds      1000,00h
schl2:  dw      12
        dw      10
        dw      60
        dw      1000
schl1:  dw      12
spos:   dw      10
lsl:    dw      60
litab:  dw      4096
itab:   ds      4*1024,00h
schls:  db      "101253000000"
logs:   ds      60,00h
```

```
        db      "x"
amld:   db      "TEST KISAMC  *** KEY/EXTEND MIT "
        db      "ZWEI DATEIEN ****"
zwsp:   ds      2
zw2:    ds      2
alogs:  dw      1
        dw      logs
        dw      60
konst:  db      "RUECKKEHRCODE: "
rcb:    db      00h,00h
        db      24h
stack:  ds      50,00h          ;Stack
stkptr: dw      0               ;Rettebereich System-Stack

        ;UP fuer Generierung Schluessel

slup:   push    hl
        .
        .
        Siehe Testprogramm KEY-Funktion!
        .
        .
        ret

        ;UP FECO - Ausgabe Fehlercode

feco:   ld      a,(dnlw)
        .
        .
        Siehe Testprogramm EXTEND-Funktion!
        .
        .
        ret

sutb:   db      00h,30h,30h
        db      02h,30h,32h
        .
        .
        Siehe Testprogramm EXTEND-Funktion!
        .
        .
        end
```

2. Beispiele fuer aufrufende BASIC-Programme

2.1. Besonderheiten

Die Zeichenkettenverarbeitung im BASIC-System BASC 1520 (SCPX) erlaubt keine einfache Uebergabe einer Zeichenkettenadresse. Das BASIC-System generiert fuer Zeichenketten Zeiger, die auf ein Drei-Byte-Element verweisen. In diesem Tripel befindet sich im ersten Byte die Laenge der Zeichenkette und im 2. u. 3. Byte die Adresse der Zeichenkette. Dieser Fakt erfordert eine gesonderte Uebernahme aller Zeichenkettenparameter (erster Parameter in allen Funktionen, sowie Suchschluessel und Zeichenketten-Satzelemente). Zur Sicherung einer korrekten Parameteruebernahme ist fuer rufende BASIC-Programme in KISAMC der spezielle Eintrittspunkt **BKISAM** erforderlich.

2.2. Allgemeine Hinweise fuer die Definition und Uebergabe von Parametern

1. Ein zu lesender bzw. schreibender Datensatz wird immer ueber die Definition eines INTEGER-Feldes beschrieben (siehe Abschnitt 5.3.2). Die fuer das Feld definierte Anfangsadresse ist an KISAMC als Parameter zu uebergeben. In BASIC wird eine Feldanfangsadresse durch die Angabe des ersten Feldelementes (Index 0) uebergeben.

   Beispiel:

   ```
   DIM  FELD%(10)       - Feldvereinbarung
   CALL UPX(FELD%(0)) - Uebergabe der Anfangsadresse des
                        Feldes FELD%
   ```

   Dies trifft fuer die Uebergabe der Adresse der Indextabelle ebenso zu, die als INTEGER-Feld zu definieren ist, sowie auch fuer den dritten Parameter der OPEN-Funktion.

2. Der Schluessel und der erste Parameter fuer jede KISAMC-Funktion sind als Zeichenkettenvariable zu vereinbaren, da diese generell ASCII-Zeichen sind.

3. Sind Datensatzelemente Zeichenkettenwerte, muss deren Adresse im INTEGER-Feld zur Beschreibung des Datensatzes nach jeder Aenderung des Zeichenkettenwertes neu eingetragen werden. Dieser Fakt ist in der internen Struktur des BASIC-Systems begruendet, das bei jeder Zeichenketten-Wertzuweisung in dem genannten Drei-Byte-Element eine neue Adresse fuer die Kette generiert. Da KISAMC ueber die Art der Satzelemente keine Informationen hat, muss der BASIC-Nutzer fuer die richtige Adressuebergabe sorgen.
   Fuer die Generierung von Schluesseln beim Erstellen einer Datei ist dies zum Beispiel erforderlich (siehe Beispielprogramm **BEXTEND.BAS**!).
   Die Adresse einer Zeichenkette wird generell wie folgt ermittelt:

```
ADR%= 256*PEEK(VARPTR(zeichenkette)+2)+
      PEEK(VARPTR(zeichenkette)+1)
```

4. Die Adressen der Datensatzelemente (siehe Abschnitt 5.2.3.) werden wie folgt bereitgestellt:

   a) fuer Zeichenkettenvariable:

```
A%(i)= 256*PEEK(VARPTR(zeichenkette))+2)+
       PEEK(VARPTR(zeichenkette))+1)
```

   A% sei das INTEGER-Feld zur Beschreibung des Datensatzes

   b) fuer INTEGER- und REAL-Werte:

```
A%(i)= VARPTR(variable/konstante)
```

2.3. Parameteraufbereitung und -uebergabe

2.3.1. OPEN-Funktion

Parameter siehe 1.1. der Anlage!

Realisierung:

1) Definition Zeichenkette (HL-Operand)

   Beispiel:

```
DN$= "B:DAT.TST"       - fester Teil der Kette
LWD$= CHR$(1)+DN$      - Kombination mit Funktionscode
```

2) Definition Indextabelle (DE-Operand)

   Beispiel:

```
DIM ITAB%(500)         - Indextabelle 1000 Byte lang
```

3) Definition des INTEGER-Feldes fuer Schluessellaenge, Schluesselposition, logische Satzlaenge, Laenge Indextabelle (BC-Operand)

   Beispiel:

```
DIM PAR%(3)            - INTEGER-Feld mit 4 Elementen
PAR%(0)=12             - Schluessellaenge
PAR%(1)=16             - Schluesselposition
PAR%(2)=42             - logische Satzlaenge
PAR%(3)=1000           - Laenge Indextabelle
```

4) Aufruf der OPEN-Funktion

   Beispiel:

```
CALL BKISAM (LWD$,ITAB%(0),PAR%(0))
```

Bemerkung: ITAB%(0) und PAR%(0) bezeichnen die Anfangsadressen der INTEGER-Felder ITAB% und PAR%. Mit LWD% wird der Zeiger auf das Drei-Byte-Element uebergeben.

2.3.2. CLOSE-Funktion

Parameter siehe 1.6. der Anlage!

Realisierung:

1) Definition Zeichenkette (HL-Operand)

   Beispiel:

```
DN$= "B:DAT.TST"        - fester Teil der Kette
LWD%= CHR$(6)+DN$       - Kombination mit Funktionscode
```

2) Aufruf CLOSE-Funktion

   Beispiel:

```
CALL BKISAM (LWD%)
```

2.3.3. KEY-Funktion

Parameter siehe 1.2. der Anlage!

Realisierung:

1) Definition Zeichenkette (HL-Operand)

   Beispiel:

```
DN$= "B:DAT.TST"        - fester Teil der Kette
LWD%= CHR$(2)+DN$       - Kombination mit Funktionscode
```

2) Definition Datensatz bzw. Datensatzelemente

   Beispiel:

```
S1$= SPACE$(15)         - Satzelement 1 (Zeichenkette)
SCHL$= SPACE$(12)       - Satzelement 2 (Schluessel)
S2$= SPACE$(15)         - Satzelement 3 (Zeichenkette)
```

3) Definition INTEGER-Feld zur Beschreibung des Datensatzes

   Beispiel:

```
DIM FELD%(6)            - INTEGER-Feld mit 7 Elementen

FELD%(0)= 3             - Anzahl Datensatzelemente =3
```

```
FELD%(1)= 256*PEEK(VARPTR(S1$)+2)+PEEK(VARPTR(S1$)+1)
                        - Adresse Satzelement 1
FELD%(2)= 15            - Laenge Satzelement 1

FELD%(3)= 256*PEEK(VARPTR(SCHL$)+2)+PEEK(VARPTR(SCHL$)+1)
                        - Adresse Satzelement 2
FELD%(4)= 12            - Laenge Satzelement 2

FELD%(5)= 256*PEEK(VARPTR(S2$)+2)+PEEK(VARPTR(S2$)+1)
                        - Adresse Satzelement 3
FELD%(6)= 15            - Laenge Satzelement 3
```

4) Definition Suchschluessel (BC-Operand)

Beispiel:

```
SUSL$= "505-33-60020"
```

5) Aufruf KEY-Funktion

Beispiel:

```
CALL BKISAM (LWD$,FELD%(0),SUSL$)
```

2.3.4. READ-Funktion

Parameter siehe 1.3. der Anlage!

Realisierung:

1) Definition Zeichenkette (HL-Operand)

Beispiel:

```
DN$= "B:DAT.TST"        - fester Teil der Kette
LWD$= CHR$(5)+DN$       - Kombination mit Funktionscode
```

2) Definition Datensatz bzw. Datensatzelemente

Beispiel:

```
S$= SPACE$(42)          -  zusammmenhaengender Datensatz
                           = 1 Datensatzelement
```

3) Definition INTEGER-Feld zur Beschreibung des Datensatzes

Beispiel:

```
DIM F%(2)               - INTEGER-Feld mit 3 Elementen

F%(0)= 1                - Anzahl Datensatzelemente =1

F%(1)= 256*PEEK(VARPTR(S$)+2)+PEEK(VARPTR(S$)+1)
                        - Adresse Datensatz
F%(2)= 42               - Laenge Datensatz
```

4) Aufruf READ-Funktion

Beispiel:

```
CALL BKISAM (LWD%,F%(0))
```

2.3.5. EXTEND-Funktion

Parameter siehe 1.4. der Anlage!

Realisierung:

1) Definition Zeichenkette (HL-Operand)

Beispiel:

```
DN%= "B:DAT.TST"        - fester Teil der Kette
LWD%= CHR%(4)+DN%       - Kombination mit Funktionscode
```

2) Definition Datensatz bzw. Datensatzelemente

Beispiel:

```
S1%="xxxxxxxxxxxxxxx" - Satzelement 1 (Zeichenkette)
SUSL%="505-33-60001"  - Satzelement 2 (Schluessel)
S2%="zzzzzzzzzzzzzzz" - Satzelement 3 (Zeichenkette)
```

3) Definition INTEGER-Feld zur Beschreibung des Datensatzes

Beispiel:

```
DIM SATZ%(6)              - INTEGER-Feld mit 7 Elementen

SATZ%(0)= 3               - Anzahl Datensatzelemente =3

SATZ%(1)= 256*PEEK(VARPTR(S1%)+2)+PEEK(VARPTR(S1%)+1)
                          - Adresse Satzelement 1
SATZ%(2)= 15              - Laenge Satzelement 1

SATZ%(3)= 256*PEEK(VARPTR(SUSL%)+2)+PEEK(VARPTR(SUSL%)+1)
                          - Adresse Satzelement 2
SATZ%(4)= 12              - Laenge Satzelement 2

SATZ%(5)= 256*PEEK(VARPTR(S2%)+2)+PEEK(VARPTR(S2%)+1)
                          - Adresse Satzelement 3
SATZ%(6)= 15              - Laenge Satzelement 3
```

4) Aufruf der EXTEND-Funktion

Beispiel:

```
CALL BKISAM (LWD%,SATZ%(0))
```

2.3.6. UPDATE-Funktion

Parameter siehe 1.5. der Anlage!

Realisierung:

Die UPDATE-Funktion kann nur nach erfolgtem READ oder KEY ausgefuehrt werden, da sie sich auf einen gelesenen Satz bezieht. Die Aufbereitung und Vermittlung der Parameter ist analog zur EXTEND-Funktion (2.3.5. der Anlage). Der Funktionscode muss entsprechend der UPDATE-Funkton eingestellt sein (CHR$(3)=03H).

```
10 ' ***              PROGRAMM BEXTEND                ***
20 ' *** TEST EXTEND-FUNKTION KISAMC FUER BASIC ***
30 '
40 ' Die  Datei  "DAT.TST" (Laufwerk B) wird  neu  angelegt.Es
50 ' werden  im  Zyklus  insgesamt 100 Saetze  der  Laenge  42
60 ' aufgezeichnet. Intern  wird ein Schluessel  generiert, um
70 ' jedem  Datensatz seinen spezifischen Schluessel  zuzuord-
80 ' nen.
90 '
100 '
110 ' Definition Indextabelle:
120 DIM ITAB%(500)
130 ' Definition ersten Parameter fuer alle Funktionen
140 ' (Laufwerk, Dateibezeichnung):
150 DN$="B:DAT.TST"
160 ' Funktionscode OPEN-Funktion ketten mit "DN$":
170 LWD$=CHR$(1)+DN$
180 ' Definition festen Teil des Suchschluessels:
190 SCHL1$="505-33-6"
200 A=0
210 ' Satzelement 1:
220 S1$="aaaaaaaaaaaaaaa"
230 ' Satzelement 3:
240 S2$="bbbbbbbbbbbbbbb"
250 ' INTEGER-Feld zur Beschreibung des Datensatzaufbaues:
260 DIM SATZ%(6)
270 ' Anzahl Satzelemente:
280 SATZ%(0)=3
290 ' Adresse Satzelement 1:
300 SATZ%(1)=256*PEEK(VARPTR(S1$)+2)+PEEK(VARPTR(S1$)+1)
310 ' Laenge Satzelement 1:
320 SATZ%(2)=15
330 ' Adresse Satzelement 2 (Schluessel):
340 SATZ%(3)=256*PEEK(VARPTR(SCHL$)+2)+PEEK(VARPTR(SCHL$)+1)
350 ' Laenge Satzelement 2 (Schluessel):
360 SATZ%(4)=12
370 ' Adresse Satzelement 3:
380 SATZ%(5)=256*PEEK(VARPTR(S2$)+2)+PEEK(VARPTR(S2$)+1)
390 ' Laenge Satzelement 3:
400 SATZ%(6)=15:
410 ' INTEGER-Feld fuer Schluessellaenge, Schluesselposition,
420 ' logische Satzlaenge, Laenge Indextabelle:
430 DIM PAR%(3)
440 ' Schluessellaenge:
450 PAR%(0)=12
460 ' Schluesselpositon:
470 PAR%(1)=16
480 ' logische Satzlaenge:
490 PAR%(2)=42
500 ' Laenge Indextabelle:
510 PAR%(3)=1000
520 PRINT "EXTEND-Funktion BASIC":PRINT
530 '
540 ' *** OPEN-Funktion ***
550 '
560 CALL BKISAM(LWD$,ITAB%(0),PAR%(0))
```

```
570 ' Test Rueckkehrcode:
580 IF LEFT$(LWD$,1)=CHR$(0) THEN PRINT "RC: 00h":GOTO 610
590 IF LEFT$(LWD$,1)=CHR$(24) THEN PRINT "RC: 18h"
    ELSE PRINT "RC: ";ASC(LEFT$(LWD$,1)):PRINT "ENDE":END
600 ' EXTEND-Schleife:
610 FOR I=1 TO 100
620 A=A+1
630 ' Schluesselgenerierung:
640 SCHL2$=STRING$(5-LEN(STR$(A)),48)+
    RIGHT$(STR$(A),LEN(STR$(A))-1)
650 SCHL$=SCHL1$+SCHL2$
660 ' Adresse Schluessel im  INTEGER-Feld "SATZ" regenerieren:
670 SATZ%(3)=256*PEEK(VARPTR(SCHL$)+2)+PEEK(VARPTR(SCHL$)+1)
680 ' Funktionscode EXTEND:
690 LWD$=CHR$(4)+DN$
700 '
710 ' *** EXTEND-Funktion ***
720 '
730 CALL BKISAM(LWD$,SATZ%(0))
740 ' Test Rueckkehrcode:
750 IF LEFT$(LWD$,1)=CHR$(0) OR LEFT$(LWD$,1)=CHR$(24)
    THEN PRINT "EXTEND O.K."
    ELSE PRINT "RC: ";ASC(LEFT$(LWD$,1)):GOTO 770
760 NEXT I
770 PRINT "ENDE PROGRAMM"
780 ' Funktionscode CLOSE:
790 LWD$=CHR$(6)+DN$
800 '
810 ' *** CLOSE-Funktion ***
820 '
830 CALL BKISAM(LWD$)
840 END
```

```
10 ' ***              PROGRAMM BASKIS                    ***
20 ' *** TEST KEY/UPDATE UND READ KISAMC FUER BASIC ***
30 '
40 '
50 ' Aus  der Datei "DAT.TST" (Laufwerk B) werden durch  wahl-
60 ' weise  ueber Tastatur einzugebende Suchschluessel  Saetze
70 ' gelesen.  Die  gefundenen Seatze  werden  angezeigt.  Bei
80 ' nicht  gefundenen  Saetzen erfolgt Mitteilung des  Rueck-
90 ' kehrcodes OCH, dezimal angezeigt mit "RC: 12". Die gefun-
100' denen Saetze werden mit neuem Inhalt belegt und ueber die
110' Funktion  UPDATE zurueckgeschrieben.  Durch  Eingabe  der
120' Endekennung  "E"  wird das zyklische Ansteuern  der  KEY-
130' Funktion  beendet.  Danach erfolgt automatisch Lesen  und
140' Anzeige aller Saetze der Datei (Funktion READ).  Das Pro-
150' gramm endet mit der EOF-Meldung (Rueckkehrcode 17H).
160 '
170 ' Definition Indextabelle:
180 DIM ITAB%(500)
190 ' Definition  ersten Parameter fuer alle Funktionen (Funk-
200 ' tionscode, Laufwerk, Dateibezeichnung):
210 DN$="B:DAT.TST"
220 LWD$=CHR$(1)+DN$
230 A=0
240 ' Satzelement 1:
250 S1$="ccccccccccccccc"
260 ' Satzelement 3:
270 S2$="ddddddddddddddd"
280 ' INTEGER-Feld zur Beschreibung des Datensatzaufbaues fuer
290 ' den ueber UPDATE rueckzuschreibenden Satz:
300 DIM SATZ%(6)
310 ' Anzahl Satzelemente fuer UPDATE:
320 SATZ%(0)=3
330 ' Adresse Satzelement 1 fuer UPDATE:
340 SATZ%(1)=256*PEEK(VARPTR(S1$)+2)+PEEK(VARPTR(S1$)+1)
350 ' Laenge Satzelement 1 UPDATE:
360 SATZ%(2)=15
370 ' Adresse Satzelement 2 (Schluessel) fuer UPDATE:
380 SATZ%(3)=256*PEEK(VARPTR(SUSL$)+2)+PEEK(VARPTR(SUSL$)+1)
390 ' Laenge Satzelement 2 (Schluessel) fuer UPDATE:
400 SATZ%(4)=12
410 ' Adresse Satzelement 3 fuer UPDATE:
420 SATZ%(5)=256*PEEK(VARPTR(S2$)+2)+PEEK(VARPTR(S2$)+1)
430 ' Laenge Satzelement 3 fuer UPDATE:
440 SATZ%(6)=15
450 ' Einlesebereich Datensatz fuer KEY:
460 STZ2$=SPACE$(42)
470 ' Integerbereich zur Beschreibung  des  mit KEY zu suchen-
480 ' den Satzes:
490 DIM V%(2)
500 ' Anzahl Satzelemente =1:
510 V%(0)=1
520 ' Adresse Einlesebereich Datensatz bei KEY und READ:
530 V%(1)=256*PEEK(VARPTR(STZ2$)+2)+PEEK(VARPTR(STZ2$)+1)
540 ' Laenge Einlesebereich fuer Datensatz bei KEY und READ:
550 V%(2)=42
560 ' INTEGER-Feld  fuer Schluessellaenge, Schluesselposition,
```

```
570 ' logische Satzlaenge, Laenge Indextabelle (3. Par. OPEN):
580 DIM PAR%(3)
590 ' Schluessellaenge:
600 PAR%(0)=12
610 ' Schluesselposition:
620 PAR%(1)=16
630 ' logische Satzlaenge:
640 PAR%(2)=42
650 ' Laenge Indextabelle:
660 PAR%(3)=1000
670 ' Bereich fuer Suchschluessel:
680 SSL$=SPACE$(12)
690 '
700 ' *** OPEN-Funktion ***
710 '
720 CALL BKISAM(LWD$,ITAB%(0),PAR%(0))
730 ' Test Rueckkehrcode:
740 IF LEFT$(LWD$,1)=CHR$(0) THEN PRINT "RC: 00h":GOTO 760
750 IF LEFT$(LWD$,1)=CHR$(24) THEN PRINT "RC: 18h"
    ELSE PRINT "Fehler OPEN, RC: ";ASC(LEFT$(LWD$,1)):END
760 PRINT "KEY-/UPDATE-Funktion":PRINT
770 EIN$=INPUT$(1):IF EIN$="E" THEN 1000
780 PRINT "Suchschluessel: ";
790 ' Eingabe Suchschluessel:
800 LINE INPUT SSL$
810 SUSL$=LEFT$(SSL$,12)
820 ' Adresse Schluessel im  INTEGER-Feld "SATZ" regenerieren:
830 SATZ%(3)=256*PEEK(VARPTR(SUSL$)+2)+PEEK(VARPTR(SUSL$)+1)
840 ' Funktionscode KEY:
850 LWD$=CHR$(2)+DN$
860 '
870 ' *** KEY-Funktion ***
880 '
890 CALL BKISAM(LWD$,V%(0),SUSL$)
900 ' Test Rueckkehrcode:
910 IF LEFT$(LWD$,1)=CHR$(0) THEN PRINT
    "RC: 00h , gefundener Satz:   "; STZ2$:GOTO 930
    ELSE PRINT "RC: ";ASC(LEFT$(LWD$,1)):GOTO 770
920 ' Funktionscode UPDATE:
930 LWD$=CHR$(3)+DN$
940 '
950 ' *** UPDATE-Funktion ***
960 '
970 CALL BKISAM(LWD$,SATZ%(0))
980 'Test Rueckkehrcode:
990 IF LEFT$(LWD$,1)=CHR$(0) THEN PRINT
    "RC: 00h, geschriebener Satz: ";S1$;SUSL$;S2$:GOTO 770
    ELSE PRINT "RC: ";ASC(LEFT$(LWD$,1)):GOTO 1230
1000 PRINT "READ-Funktion":PRINT
1010 ' Funktionscode CLOSE:
1020 LWD$=CHR$(6)+DN$
1030 '
1040 ' *** CLOSE-Funktion ***
1050 '
1060 ' CLOSE, weil Datei ab Dateianfang gelesen werden soll.
1070 CALL BKISAM(LWD$)
```

```
1080 ' Funktionscode OPEN:
1090 LWD%=CHR%(1)+DN%
1100 '
1110 ' *** OPEN-Funktion ***
1120 '
1130 CALL BKISAM(LWD%,ITAB%(0),PAR%(0))
1140 ' Funktionscode READ:
1150 LWD%=CHR%(5)+DN%
1160 '
1170 ' *** READ-Funktion ***
1180 '
1190 CALL BKISAM(LWD%,V%(0))
1200 ' Test Rueckkehrcode auf EOF (17H):
1210 IF LEFT%(LWD%,1)=CHR%(23) THEN
     PRINT "RC: 17h, Dateiende!":GOTO 1230
     ELSE IF LEFT%(LWD%,1)=CHR%(0) THEN
     PRINT "gelesener Satz: ";STZ2%:GOTO 1150
     ELSE PRINT "RC: ";ASC(LEFT%(LWD%,1))
1220 ' Funktioncode CLOSE:
1230 LWD%=CHR%(6)+DN%
1240 '
1250 ' *** CLOSE-Funktion ***
1260 '
1270 CALL BKISAM(LWD%)
1280 PRINT "ENDE PROGRAMM"
1290 END
```

3. Beispiele fuer aufrufende FORTRAN-Programme

3.1 Allgemeine Hinweise

Zur Definition der einzelnen Parameter werden Felder vereinbart.
Die Uebergabe der Parameter erfolgt bei Aufruf von KISAMC ueber die Angabe des ersten Feldelementes.

3.2 Beispiel fuer Test der EXTEND-Funktion

Das nachfolgende Programm **INDEX** ist analog dem BASIC-Programm BEXTEND unter Abschnitt 2 dieser Anlage:

```
c Test EXTEND-Funktion KISAMC fuer FORTRAN
c INDEX.FOR
c
        integer itab,satz,par
        byte name,schl,s1,s2
        dimension itab(500)
        dimension name(15)
        dimension schl(12)
        dimension s1(15)
        dimension s2(15)
        dimension satz(7)
        dimension par(4)
c
        data name/x'01','B',':','D','A','T','.','T','S','T'/
        data schl/'5','0','5','-','3','3','-','6','0','0',
                  '0','0'/
        data s1/15*'a'/
        data s2/15*'b'/
c
c Anzahl Satzelemente:
        satz(1)=3
c Adresse Satzelement 1:
        satz(2)=x'051d'
c Laenge Satzelement 1:
        satz(3)=15
c Adresse Satzelement 2:
        satz(4)=x'0511'
c Laenge Satzelement 2:
        satz(5)=12
c Adresse Satzelement 3:
        satz(6)=x'052c'
c Laenge Satzelement 3:
        satz(7)=15
c
c Schluessellaenge:
        par(1)=12
c Schluesselposition:
        par(2)=16
c logische Satzlaenge:
        par(3)=42
```

```
c Laenge Indextabelle:
        par(4)=1000
c
c OPEN-Funktion
c
        call kisamc(name(1),itab(1),par(1))
c
c Test Rueckkehrcode:
        if (name(1).eq.x'00') write (1,50)
 50     format(1x,'RC=00h')
        if (name(1).eq.x'18') write (1,60)
 60     format(1x,'RC=18h')
        if (name(1).eq.x'0b') goto 110
        goto 120
 110    write (1,130)
 130    format(1x,'Unsortierte Datei')
        goto 140
c
c EXTEND-Schleife
c
 120    do 1 i=1,9
           schl(12)=schl(12)+1
           name(1)=x'04'
c
c EXTEND-Funktion
c
           call kisamc(name(1),satz(1))
c Test Rueckkehrcode:
           if ((name(1).eq.x'00').or.(name(1).eq.x'18'))
           goto 80
           write(1,90)
 90        format(1x,'RC()00h oder 18h')
           goto 1
 80        write (1,70)
 70        format(1x,'EXTEND o.k.')
 1      continue
        write (1,20)
 20     format(1x,'Ende Programm')
c
c Funktionscode CLOSE:
        name(1)=x'06'
c
c CLOSE-Funktion
c
        call kisamc(name(1))
 140    end
```

3.3 Beispiel fuer Test der KEY-, UPDATE- und READ-Funktion

Das nachfolgende Programm **KINDEX** ist analog dem BASIC-Programm BASKIS unter Abschnitt 2 dieser Anlage:

```
c Test KEY/UPDATE und READ KISAMC fuer FORTRAN
c KINDEX.FOR
c
        integer itab,satz,v,par
        byte name,s1,s2,stz2,ssl,ein
        dimension itab(500)
        dimension s1(15)
        dimension s2(15)
        dimension name(15)
        dimension satz(7)
        dimension stz2(42)
        dimension v(3)
        dimension par(4)
        dimension ssl(12)
c
        data name/x'01','B',':','D','A','T','.','T','S','T'/
        data s1/15*'c'/
        data s2/15*'d'/
c
c Anzahl Satzelemente fuer UPDATE:
        satz(1)=3
c Adresse Satzelement 1 fuer UPDATE:
        satz(2)=x'0517'
c Laenge Satzelement 1 fuer UPDATE:
        satz(3)=15
c Adresse Satzelement 2 (Schluessel) fuer UPDATE:
        satz(4)=x'055f'
c Laenge Satzelement 2 (Schluessel) fuer UPDATE:
        satz(5)=12
c Adresse Satzelement 3 fuer UPDATE:
        satz(6)=x'0526'
c Laenge Satzelement 3 fuer UPDATE:
        satz(7)=15
c INTEGER-Feld zur Beschreibung des mit KEY zu suchenden
  Satzes:
c Anzahl Satzelemente=1:
        v(1)=1
c Adresse Einlesebereich Datensatz mit KEY und READ:
        v(2)=x'0535'
c Laenge Einlesebereich fuer Datensatz bei KEY und READ:
        v(3)=42
c INTEGER-Feld fuer Schluessellaenge, Schluesselposition,
c logische Satzlaenge, Laenge Indextabelle:
c Schluessellaenge:
        par(1)=12
c Schluesselposition:
        par(2)=16
c log. Satzlaenge:
        par(3)=42
c Laenge Indextabelle:
```

```
        par(4)=1000
c
c OPEN-Funktion
c
        call kisamc(name(1),itab(1),par(1))
c Test Rueckkehrcode:
c
        if (name(1).eq.x'00') write (1,10)
 10     format(1x,'RC=00h')
        if (name(1).eq.x'18') write (1,20)
 20     format(1x,'RC=18h')
        if ((name(1).ne.x'00').and.(name(1).ne.x'18'))
        goto 100
c
        write (1,40)
 40     format(1x,'"KEY-/UPDATE-Funktion"')
 90     read (1,50)ein
 50     format(a1)
        if (ein.eq.x'45') goto 500
        write (1,60)
 60     format(1x,'Eingabe Suchschluessel: ')
        read (1,70)ssl
 70     format(12a1)
c
c Funktion KEY
c
        name(1)=x'02'
        call kisamc(name(1),v(1),ssl(1))
c
c Test Rueckkehrcode:
c
        if (name(1).eq.x'00') write (1,80)stz2
 80     format(1x,'RC=00h',42a1)
        if (name(1).ne.x'00') goto 90
c
c Funktionscode UPDATE:
c
        name(1)=x'03'
        call kisamc(name(1),satz(1))
c
c Test Rueckkehrcode:
        if (name(1).eq.x'00') write (1,110)s1,ssl,s2
 110    format(1x,'RC=00h , geschriebener Satz:
        ',15a1,12a1,15a1)
        if (name(1).ne.x'00') goto 200
        goto 90
 500    write (1,120)
 120    format(1x,'READ-Funktion',1x)
c
c CLOSE-Funktion
c
        name(1)=x'06'
        call kisamc(name(1))
c OPEN-Funktion
        name(1)=x'01'
        call kisamc(name(1),itab(1),par(1))
```

```
c Funktionscode READ:
 360     name(1)=x'05'
         call kisamc(name(1),v(1))
c Test Rueckkehrcode:
        if (name(1).eq.x'17') goto 300
         if (name(1).eq.x'00') goto 350
c
c Funktionscode CLOSE:
 200     name(1)=x'06'
         call kisamc(name(1))
         write (1,150)
 150     format(1x,'Ende Programm')
         goto 400
 100     write (1,30)
 30      format(1x,'Fehler OPEN')
         goto 400
 300     write (1,130)
 130     format(1x,'RC=17h, Dateiende!')
         goto 200
 350     write (1,140)stz2
 140     format(1x,'gelesener Satz:  ',42a1)
         goto 360
 400     end
```

4. Beispiele fuer aufrufende C-Programme

3.1 Besonderheiten

KISAMC kann im C-System CC1520 (SCPX) abgearbeitet werden. Zur Sicherung einer korrekten Parameteruebernahme ist fuer rufende C-Programme der spezielle Eintrittspunkt CKISAM erforderlich.

3.2 Realisierung

KISAMC wird in einem C-Programm durch den Funktionsnamen

```
ckisam(par.1[,par.2[,par.3]])
```

aufgerufen. Die Vereinbarung der Parameter ergibt sich aus dem nachfolgenden Beispielprogramm.
Das C-Programm wird compiliert und assembliert. Nachfolgend wird es mit dem Kommando

```
LINK CHDR,<dateiname>,KISAMC,CTRLIB/S,<dateiname>/N/E
```

zu einem abarbeitbaren Programm verbunden.

3.3 Beispiel fuer Aufruf aller KISAMC-Funktionen ueber ein C-Programm

```
/* Test indexsequentieller Zugriff im C-System
 * CKISAMC :
 * Parameteruebergabe und Aufruf von KISAMC
 */

char par_1[15] = "*B:DAT.TST"; /* Name Testdatei */

/* Rahmenprogramm */
main()
{
char cmd;       /* fuer Kdo.-Eingabe */
char *open(), *close();
char *extend();
char *readds(), *key();

write(2, "\f***** INDEXSEQUENTIELLER ZUGRIFF *****\n", 42);
printf("\n××× Testdatei: %p\n", par_1); /*Ausgabe Dateiname*/
printf("\n××× Adresse FC: %h \t Inhalt FC: %h\n", \
        &par_1[0], par_1[0]);

/* Kommandoeingabe */

do      {
        write(2, "\33\206\200\24", 4); /* Kursorposit. */
        write(2, "\n??? Eingabe Kommando:   ", 25);
        scanf("%1p", &cmd);
```

```
        switch(cmd) {
             case 'o': par_1[0] = '\01'; open(par_1); break;
             case 'k': par_1[0] = '\02'; key(par_1); break;
             case 'u': par_1[0] = '\03'; extend(par_1); break;
             case 'e': par_1[0] = '\04'; extend(par_1); break;
             case 'r': par_1[0] = '\05'; readds(par_1); break;
             case 'c': par_1[0] = '\06'; close(par_1);
                  };
        printf("\n*** Adresse RC: %h \t Inhalt RC: %h\n", \
                &par_1[0], par_1[0]);
        write(2, "\n??? Neue Funktion? (j/n) ", 26);
        scanf("%1p", &cmd);
        } while(cmd != 'n');

write(2, "\n***** PROGRAMMENDE *****\n", 28);
}

/* OPEN - FKT. */
char *open(par_1)
char *par_1;    /* Verweis auf Funktionscode,Dateiname */
{
int par_2[500]; /* Indextabelle */
int par_3[4];   /* Bechreibung Dateiaufbau */
char *ckisam();
par_3[0] = 6;   /* Schluessellaenge in Byte */
par_3[1] = 1;   /* Schluesselposition */
par_3[2] = 26;  /* logische Satzlaenge */
par_3[3] = 1000; /* Laenge Indextabelle in Byte */

write(2, "\n***** OPEN *****\n", 18);
ckisam(par_1, par_2, par_3);
write(2, "*** END OPEN ***\n", 17);
return(par_1);
}

/* CLOSE - FKT. */
char *close(par_1)
char *par_1;
{
char *ckisam();

write(2, "\n***** CLOSE *****\n", 19);
ckisam(par_1);
write(2, "*** END CLOSE ***\n", 18);
return(par_1);
}

struct ds {                         /* Aufbau Datensatz */
        char schl_nr[6];            /* Schluesselbegriff */
        char bez[10];               /* Artikelbezeichnung */
        int stck;                   /* Stueckzahl */
        double evp;                 /* Stueckpreis */
          };
```

```
/* EXTEND/UPDATE - FKT. */
char *extend(par_1)
char *par_1;
{
int par_2[3];     /* Beschreibung Datensatz */
struct ds satz;   /* Speicherplatz fuer Datensatz */
char *ckisam();

par_2[0] = 1;    /* Anzahl der Datensatzelemente */
par_2[1] = &satz; /* Adresse Datensatz */
par_2[2] = 26;   /* logische Satzlaenge */

write(2, "\n??? Eingabe Daten:\n", 20);
write(2, "\tLfd. Nummer:  ", 15);
scanf("%p", satz.schl_nr);
write(2, "\tArtikel:      ", 15);
scanf("%p", satz.bez);
write(2, "\tStueckzahl:   ", 15);
scanf("%i", satz.stck);
write(2, "\tStueckpreis:  ", 15);
scanf("%d", satz.evp);
if(par_1[0] == '\04')
        {
        write(2, "\n***** EXTEND *****\n", 20);
        ckisam(par_1, par_2);
        write(2, "*** END EXTEND ***\n", 19);
        }
else    {
        write(2, "\n***** UPDATE *****\n", 20);
        ckisam(par_1, par_2);
        write(2, "*** END UPDATE ***\n", 19);
        }
return(par_1);
}
/* KEY - FKT. */
char *key(par_1)
char *par_1;
{
int par_2[3];     /* Beschreibung Datensatz */
char schl[6];     /* Schluesselbegriff */
struct ds satz;   /* Speicherplatz fuer Datensatz */
char *ckisam();

par_2[0] = 1;    /* Anzahl der Datensatzelemente */
par_2[1] = &satz; /* Adresse Datensatz */
par_2[2] = 26;   /* logische Satzlaenge */

write(2, "\n??? Eingabe Schl.-Nr.:  ", 25);
scanf("%p", schl);
write(2, "\n***** KEY *****\n", 17);
ckisam(par_1, par_2, schl);
write(2, "*** END KEY ***\n", 16);
if(par_1[0] == '\0')
        printf("\nArtikel: %p\n", satz.bez);
return(par_1);
}
```

```
/* READ - FKT. */
char *readds(par_1)
char *par_1;
{
int par_2[3];     /* Beschreibung Datensatz */
struct ds satz;   /* Speicherplatz fuer Datensatz */
char *ckisam();

par_2[0] = 1;    /* Anzahl der Datensatzelemente */
par_2[1] = &satz; /* Adresse Datensatz */
par_2[2] = 26;   /* logische Satzlaenge */

write(2, "\n***** READ *****\n", 18);
ckisam(par_1, par_2);
write(2, "*** END READ ***\n", 17);
if(par_1[0] == '\0')
        printf("\nArtikel: %p\n", satz.bez);
return(par_1);
}
```